jc560 U.S. PTO
09/003972
01/07/98

APPENDIX B

Inventor: Everett W. Stoub

Title: System for Converting Scrolling Display To Non-Scrolling Columnar Display

BEST AVAILABLE COPY

```
/**
 *  %W% %E% Everett Stoub
 *
 *  Copyright (c) '96-'97 ION Web Products, Inc. All Rights Reserved.
 *
 *  ez_html applet displays snaking text with in-line graphics
 *      and produces a reader screen frame when clicked
 *
 *
 *  applet param specifications
 *
 *      fileName    value=name_of_file  (required)
 *          extension must be .htm or .html for html parsing
 *          e.g. <param name="fileName" value="ezsample.htm">
 *
 *      maxColumns  value=an_integer    (optional)
 *          specifies upper limit of column number in APPLET display only
 *          e.g. <param name="maxColumns" value="2">
 *
 *      frameType   value=special_name  (optional)
 *          e.g. <param name="frameType" value="LINE">      default frameSize = 1
 *          e.g. <param name="frameType" value="FLAT">      default frameSize = 1
 *          e.g. <param name="frameType" value="RAISED">    default frameSize = 2
 *          e.g. <param name="frameType" value="DROPPED">   default frameSize = 2
 *          e.g. <param name="frameType" value="ETCHED">    default frameSize = 4
 *
 *      frameSize   value=an_integer    (optional)
 *          specifies width of optional frame
 *          e.g. <param name="frameSize" value="2">
 *
 *
 *  HTML tag parsing honors the following commands and attributes
 *
 *     • <title>...</title>             title text delimeters
 *
 *     • <base>                         document relationships
 *          href                            relative or absolute url for hyperlink
 *
 *     • <body>...</body>               body text delimeters
 *          bgcolor=color                   background color (#hhhhhh or name)
 *          text=color                      default text color
 *          link=color                      hypertext link color
 *          background=URL                  tiled applet & reader background
 *
 *     • <table>...</table>             table tag & range (replaced with hot image)
 *
 *
 *  @author Everett Stoub %I%, %G%
 */
import java.awt.*;
import java.awt.image.*;
import java.applet.*;
import java.net.*;
import java.io.*;
import java.util.*;

public class ez_html extends Applet
{
    public static Color         backColor       = new Color(239,239,239);
    public static Color         foreColor       = Color.black;
    public static Color         linkColor       = Color.blue;

    public static int           maxColumns      = 15;

    public _f_rs_db             frame           = null;
    public String               title           = null;
    private String              article         = null;
    private _p_rs_db            titlePanel      = null;
    private _p_rs_db            bodyPanel       = null;
    private _o_im_db            imageDB         = null;
    private _o_tb_db            tableDB         = null;
    private _o_fl_db            formsDB         = null;

    private Image               backImage       = null;
    private URL                 baseURL         = null;

    private MediaTracker        tracker         = null;

    private final int           NONE            = 0;
    private final int           LINE            = 1;
    private final int           FLAT            = 2;
    private final int           RAISED          = 3;
    private final int           DROPPED         = 4;
    private final int           ETCHED          = 5;

    private int                 frameType       = NONE;
    private int                 frameSize       = 0;
    private Insets              insets          = new Insets(0, 0, 0, 0);
    public final static String  CS              = "Release to create reader
screen...";
    public final static String  RS              = "Release to view reader screen...";
    public final static String  WS              = "Wait for new reader screen...";
    public final static String  ZM              = "Click to read...";

    private static final void dbg(String s)
    {
        if (debug) System.out.println(s);
    }
    private static boolean debug = false;   //  Print debugging info?

    public void disposeFrame(_f_rs_db disposee)
    {
        if (disposee != null)
        {
            Event evt = new Event(disposee, Event.WINDOW_DESTROY, null);
            Component co[] = disposee.getComponents();
            for (int i = 0; i < co.length; i++)
                co[i].handleEvent(evt);
            disposee.dispose();
        }
        if (disposee == frame)
        {
            frame = null;
        }
    }

    public Image getBackImage()
    {
        return this.backImage;
    }

    public URL getDocumentBase()
    {
        if (baseURL != null)
            return baseURL;
        else
            return super.getDocumentBase();
    }

    public _o_im_db getImageDB()
    {
        if (this.imageDB == null)
            this.imageDB = new _o_im_db(this);

        return this.imageDB;
    }

    public _o_tb_db getTableDB()
    {
        if (this.tableDB == null)
```

```
            this.tableDB = new _o_tb_db(this);

        return this.tableDB;
    }

    public _o_fl_db getFormDB()
    {
        if (this.formsDB == null)
            this.formsDB = new _o_fl_db(this);

        return this.formsDB;
    }

    public Color getBackColor()
    {
        return this.backColor;
    }

    public Color getForeColor()
    {
        return this.foreColor;
    }

    public String getTitle()
    {
        return this.title;
    }

    public MediaTracker getTracker()
    {
        if (this.tracker == null)
            this.tracker = new MediaTracker(this);

        return this.tracker;
    }

    public void init()
    {
        String s = getParameter("fileName");
        if (s != null)
        {
            article = _o_ht_db.getArticle(s, this);
        }

        String m = getParameter("maxColumns");
        if (m != null)
        {
            maxColumns = Math.max
            (   _o_tg_db.parseInt(m, maxColumns)
            ,   1
            );
        }

        this.setLayout(new BorderLayout());

        if (title != null)
        {
        //  dbg("titlePanel: "+title);

            String t = getParameter("showTitle");
            if (t != null)
            {
                if (t.toLowerCase().equals("yes"))
                {
                    titlePanel = new _p_rs_db
                    (   "<font color=\"#0000b3\"><h2 align=center>"+title
                    ,   this
                    ,   null            //  the browser is the frame
                    ,   false           //  NOT the reader screen
                    ,   1               //  fixed to center the title
                    ,   false           //  no paging indicator marks
                    ,   false           //  no page position footer
                    ,   false           //  not zoomable
                    );
                    titlePanel.textAlign = _p_rs_db.CENTER;
                    this.add("North", titlePanel);
                }
            }
        }

        bodyPanel = new _p_rs_db
        (   article
        ,   this
        ,   null                        //  the browser is the frame
        ,   false                       //  NOT the reader screen
        ,   maxColumns                  //  completely automatic
        ,   true                        //  use paging indicator marks
        ,   false                       //  no page position footer
        ,   true                        //  zoomable
        );
        this.add("Center", bodyPanel);

        String b = getParameter("frameType");
        if (b != null)
        {
            b = b.toUpperCase();
            if (b.equals("LINE"))    {   frameSize = 1;  frameType = LINE;    }
            if (b.equals("FLAT"))    {   frameSize = 1;  frameType = FLAT;    }
            if (b.equals("RAISED"))  {   frameSize = 2;  frameType = RAISED;  }
            if (b.equals("DROPPED")) {   frameSize = 2;  frameType = DROPPED; }
            if (b.equals("ETCHED"))  {   frameSize = 4;  frameType = ETCHED;  }
        }

        String w = getParameter("frameSize");
        if (w != null) try
        {
            frameSize = Math.max
            (   Integer.parseInt(w)
            ,   1
            );
        }
        catch (NumberFormatException nf) {}

        switch (frameType)
        {
            case LINE:
                insets.bottom   = frameSize;    break;
            case FLAT:
            case RAISED:
            case DROPPED:
            case ETCHED:
                insets.top      = frameSize;
                insets.left     = frameSize;
                insets.bottom   = frameSize;
                insets.right    = frameSize;
        }
    }

    public Insets insets()
    {
        return insets;
    }

    public boolean mouseDown(Event evt, int x, int y)
    {
    //  dbg("ez_html.mouseDown()");

        if (frame != null)
        {
            showStatus(RS);
        }
        else
        {
            showStatus(CS);
        }

        return true;
    }

    public boolean mouseEnter(Event evt, int x, int y)
    {
        showStatus(ZM);

        return true;
    }

    public boolean mouseExit(Event evt, int x, int y)
    {
        showStatus(_p_rs_db.MT);

        return true;
    }

    public boolean mouseMove(Event evt, int x, int y)
    {
        showStatus(ZM);

        return true;
    }

    public boolean mouseUp(Event evt, int x, int y)
    {
    //  dbg("ez_html.mouseUp()");

        if (frame != null)
        {
        //  dbg("ez_html.mouseUp(): restoring _f_rs_db");
            frame.toFront();
        }
        else
        {
        //  dbg("ez_html.mouseUp(): making new _f_rs_db");
            showStatus(WS);
            frame = new _f_rs_db
            (   title
            ,   article
            ,   this
            ,   15                          //  max columns
            ,   true                        //  present markers
            ,   true                        //  present footer
            );
        }
        return true;
    }

    public final void paint(Graphics g)
    {
    //  dbg("applet.paint()");

        int i;
        Dimension d = size();
        if (g != null) switch (frameType)
        {
            case LINE:
            {   g.setColor(getForeground());
                for (i = 0; i < frameSize; i++)
                    g.drawLine(0,d.height-i-1,d.width-i-1,d.height-i-1);
            }   break;
            case FLAT:
            {   g.setColor(getForeground());
```

```
                for (i = 0; i < frameSize; i++)
                    g.drawRect(i,i,d.width-2*i-1,d.height-2*i-1);
            }   break;
            case RAISED:
            case DROPPED:
            case ETCHED:
            {   boolean raise = frameType==RAISED;
                int R = (getForeground().getRed()+  getBackground().getRed()   )/2;
                int G = (getForeground().getGreen()+getBackground().getGreen())/2;
                int B = (getForeground().getBlue()+ getBackground().getBlue() )/2;
                g.setColor(new Color(R,G,B));
                for (i = 0; i < frameSize; i++)
                {
                    if (frameType == ETCHED) raise = i >= frameSize/2;
                    g.draw3DRect(i,i,d.width-2*i-1,d.height-2*i-1,raise);
                }
                if (frameType == ETCHED)
                {
                    g.setColor(getForeground());
                    g.drawLine(0,0,frameSize/2,frameSize/2);
                    g.drawLine(d.width-frameSize,d.height-frameSize,d.width-
frameSize/2,d.height-frameSize/2);
                    g.setColor(getBackground());
                    g.drawLine(frameSize/2,frameSize/2,frameSize,frameSize);
                    g.drawLine(d.width-frameSize/2,d.height-frameSize/2,d.width-
1,d.height-1);
                }
                else
                {
                    g.setColor((frameType==RAISED)?getBackground():getForeground());
                    g.drawLine(0,0,frameSize,frameSize);
                    g.setColor((frameType==RAISED)?getForeground():getBackground());
                    g.drawLine(d.width-frameSize,d.height-frameSize,d.width-
1,d.height-1);
                }
            }
        }
    }

    public void setBackColor(Color color)
    {
        this.backColor = color;
    }

    public void setBackImage(Image image)
    {
        this.backImage = image;
    }

    public void setBaseURL(URL baseURL)
    {
        this.baseURL = baseURL;
    }

    public void setForeColor(Color color)
    {
        this.foreColor = color;
    }

    public void setLinkColor(Color color)
    {
        this.linkColor = color;
    }

    public void setTitle(String title)
    {
        this.title = title;
    }

    public void update(Graphics g)
    {
    //  fillRect is NOT performed here to eliminate blank screen boredom during
offscreen drawing

    //  g.setColor(getBackground());
    //  g.fillRect(0, 0, width, height);
    //  g.setColor(getForeground());

        paint(g);
    }
}

/**
 *  %W% %E% Everett Stoub
 *
 *  Copyright (c) '96-'97 ION Web Products, Inc. All Rights Reserved.
 *
 *  an object to read html documents and parse head content
 *
 *  this class is not downloaded unless needed
 *
 *  @author Everett Stoub %I%, %G%
 */
import java.awt.*;
import java.io.*;
import java.net.*;
import java.util.*;

public class _o_ht_db extends Object
{
    private ez_html              ez_HTML     = null;

    private final static String alert_0      = "PLEASE READ - A Java ";
    private final static String alert_1      = "Exception has occurred. ";
    private final static String alert_2      = "Most likely, there has been a
transmission failure with one or more applet files. A Refresh or Reload command on
your browser will sometimes lead to a successful download. ";
    private final static String alert_3      = "An attempt to read a local file has
been detected! In order to read this article, you must download the applet using
hyper-text transport protocols (http://...) from a web server. Please try again using
http with a valid URL. ";
    private final static String alert_4      = "If the problem persists, please contact
the Web Master.";

    public final static String  BL           = " ";
    public final static String  BS           = "<base";
    public final static String  HF           = "href";
    public final static String  MT           = "";
    public final static String  PR           = "<p>";

    private static final void dbg(String s)
    {
        if (debug) System.out.println(s);
    }
    private static boolean debug = true;        //  Print debugging info?

    public static String getArticle(String fileName, ez_html ez_HTML)
    {
        String article = MT;

        if (fileName != null) try                                           //  new
URL, etc.
        {
            URL url = new URL(ez_HTML.getDocumentBase(), fileName);
            InputStream is = url.openStream();
            BufferedInputStream bis = new BufferedInputStream(is);

            int len = 0;
            byte buffer[] = new byte[512];
            StringBuffer pageText = new StringBuffer();
            while ((len = bis.read(buffer)) > 0)
                pageText.append(new String(buffer,0,0,len));

            bis.close();
            is.close();

            boolean isHTML = fileName.toLowerCase().endsWith(".htm")
                          || fileName.toLowerCase().endsWith(".html");      //  other
indicators would be more helpful...

            if (isHTML)                                                      //
    protect literal sections from stripping
            {
                StringBuffer articleBuffer = new StringBuffer();
                Vector pieces = new Vector(50);                              //  String
segments of pageText distinguished by filtering
                Vector status = new Vector(50);                              //
    Boolean status of each segment: true of filtering is needed

                String MCtext = pageText.toString();
                String LCtext = MCtext.toLowerCase();

            //  the plaintext tag preempts all subsequent tags

                int plaintext = LCtext.indexOf("<plaintext>");
                if (plaintext > -1)                                          //  rare:
a deprecated tag by HTML 3.2
                {
                    pieces.addElement(MCtext.substring(0,plaintext));
                    status.addElement(new Boolean(true));                    //
    filtering is needed

                    pieces.addElement(lineCanon(MCtext.substring(plaintext), false));
                    status.addElement(new Boolean(false));                   //
    filtering is complete
                }
                else
                {
                    pieces.addElement(MCtext);
                    status.addElement(new Boolean(true));                    //
    filtering is needed
                }

            //  <textarea>...</textarea> & <pre>...</pre> ranges are taken literally

                while (status.size() > 0)
                {
                    if (((Boolean)status.firstElement()).booleanValue())     //
    filtering needed
                    {
                        MCtext = (String)pieces.firstElement();
                        LCtext = MCtext.toLowerCase();

                        String TE = "</textarea>";
                        String PE = "</pre>";
                        int began, ended;
                        if ((ended = LCtext.indexOf(TE)) > -1
                        &&  (began = LCtext.lastIndexOf("<textarea",ended)) > -1)
                        {
                            ended += TE.length();                            //
    inclusive terminus

                            pieces.setElementAt(MCtext.substring(0,began),   /*
    replace original    */  0);
                            status.setElementAt(new Boolean(true),           /*
    filtering needed    */  0);

                            pieces.insertElementAt(lineCanon(MCtext.substring(began,
ended), false), 1);
```

```
                                  status.insertElementAt(new Boolean(false),       /*
    filtering complete  */  1);

                                  pieces.insertElementAt(MCtext.substring(ended),
                    2);
                                  status.insertElementAt(new Boolean(true),        /*
    filtering needed    */  2);
                              }
                              else
                              if ((ended = LCtext.indexOf(PE)) > -1
                              &&  (began = LCtext.lastIndexOf("<pre",ended)) > -1)
                              {
                                  ended += PE.length();                            //
    inclusive terminus

                                  pieces.setElementAt(MCtext.substring(0,began),   /*
    replace original    */  0);
                                  status.setElementAt(new Boolean(true),           /*
    filtering needed    */  0);

                                  pieces.insertElementAt(lineCanon(MCtext.substring(began,
ended), false), 1);
                                  status.insertElementAt(new Boolean(false),       /*
    filtering complete  */  1);

                                  pieces.insertElementAt(MCtext.substring(ended),
                    2);
                                  status.insertElementAt(new Boolean(true),        /*
    filtering needed    */  2);
                              }
                              else                                                 //   filter
it as html & tack it on
                              {
                                  articleBuffer.append(lineCanon(MCtext, true));   //
    filtering complete: tack it on

                                  pieces.removeElementAt(0);                       //   remove
original
                                  status.removeElementAt(0);                       //   remove
original
                              }
                          }
                          else                                                     //
    filtering is complete: tack it on
                          {
                              articleBuffer.append((String)pieces.firstElement());

                              pieces.removeElementAt(0);
                              status.removeElementAt(0);
                          }
                      }
                      article = htmlBodyText(articleBuffer.toString(), ez_HTML);
                  }
                  else
                  {
                      article = htmlBodyText(lineCanon(pageText.toString(), false),
ez_HTML);
                  }
              }
              catch (MalformedURLException mu)
              {
                  StringBuffer msg = new StringBuffer();
                  msg.append(alert_0).append("Malformed URL
").append(alert_1).append(alert_2).append(alert_4);

                  article = msg.toString();

                  String e = "Could not open "+fileName;
                  ez_HTML.showStatus(e);
                  System.out.println(e+": "+mu);
```

```
        }
        catch (IOException io)
        {
            StringBuffer msg = new StringBuffer();

    msg.append(alert_0).append("Input/Output").append(alert_1).append(alert_2).append(
alert_4);

            article = msg.toString();

            String e = "Could not open "+fileName;
            ez_HTML.showStatus(e);
            System.out.println(e+": "+io);
        }
        catch (SecurityException se)
        {
            StringBuffer msg = new StringBuffer();
            msg.append(alert_0).append("Security
").append(alert_1).append(alert_3).append(alert_4);

            article = msg.toString();

            String e = "Could not open "+fileName;
            ez_HTML.showStatus(e);
            System.out.println(e+": "+se);
        }
        else
        {
            StringBuffer msg = new StringBuffer();
            msg.append(alert_0).append("Input/Output").append(alert_1);
            msg.append("No article file specification was detected! ");
            msg.append(alert_2).append(alert_4);

            article = msg.toString();

            String e = "No fileName specified";
            ez_HTML.showStatus(e);
            System.out.println(e);
        }

        return article;
    }

    public static String htmlBodyText(String itsText, ez_html ez_HTML)
    {
        String newText = new String(itsText);
        String LCtext = newText.toLowerCase();

    //  look for [<!--...-->] html comments and eliminate them

        String CE = "-->";

        int began, ended = LCtext.indexOf(CE);
        while (ended > -1)                                   //  the first end-of-
comment
        {
            began = LCtext.lastIndexOf("<!--",ended);        //  its (prior) start-of-
comment
            if (began > -1)
            {
                ended += CE.length();                        //  inclusive terminus

                StringBuffer text = new StringBuffer();
                text.append(newText.substring(0,began));     //  preceeding text
                text.append(newText.substring(ended));       //  succeeding text

                newText = text.toString();
                LCtext = newText.toLowerCase();
                ended = LCtext.indexOf(CE);
            }
            else
            {
                ended = -1;                                 //  format error exit path
            }
        }

    //  look for [<title...>title text</title...]

        began = LCtext.indexOf("<title");
        if (began > -1)
        {
            began = LCtext.indexOf(">",began)+1;            //  start of title
            if (began > 0 && began < LCtext.length())
            {
                ended = LCtext.indexOf("</title",began);
                if (ended > -1)
                {
                    ez_HTML.setTitle(newText.substring(began,ended));
                }
            }
        }

        began = LCtext.indexOf(BS);
        if (began > -1)
        {
            began += BS.length();
            ended = LCtext.indexOf(">",began)+1;            //  end of base
            if (ended > 0 && ended < LCtext.length())
            {
                String itsSpec =
_o_tg_db.getTagAttribString(newText.substring(began,ended),HF);
                try
                {
                    ez_HTML.setBaseURL(new URL(ez_HTML.getDocumentBase(),itsSpec));
                }
                catch (MalformedURLException m)
                {
                    ez_HTML.setBaseURL(null);
                }
            }
        }

    //  look for [<form...>table specs</form...] recursively number document forms &
elements

        String fb = _p_rs_db.PF                + "form";
        String fe = _p_rs_db.PF + _p_rs_db.SL + "form" + _p_rs_db.SF;

        int start, formNumber = 0;

        ended = LCtext.indexOf(fe);                         //  the first end-of-form
        began = LCtext.lastIndexOf(fb, ended);              //  its (prior) start-of-
form
        start = LCtext.indexOf(_p_rs_db.SF, began);         //  its start-of-form
endpoint
        while (-1 < began && began < start && start < ended)
        {
            formNumber++;
            start += _p_rs_db.SF.length();                  //  start-of-form content
range start point

            int trail = ended + fe.length();                //  beginning of trailing
content

            String orig_tag = LCtext.substring(began,start);//  just the original form
tag
            _o_fm_db theForm = new _o_fm_db(formNumber, orig_tag);
            ez_HTML.getFormDB().newForm(theForm);

            String formContent = newText.substring(start, ended);
            String newContent = theForm.replaceContent(formContent);

            StringBuffer text = new StringBuffer();
            text.append(newText.substring(0,began))          //  preceeding text,
excluding original form tag
                .append(newContent)                          //  new form range content
(modified elements plus orig html content)
                .append(newText.substring(trail));           //  succeeding content
text, excluding original end form tag

            ended -= trail - began;                          //  this was dropped
            ended += newContent.length();                    //  this was added

            newText = text.toString();
            LCtext = newText.toLowerCase();
            ended = LCtext.indexOf(fe, ended);               //  the next end-of-form
            began = LCtext.lastIndexOf(fb, ended);           //  its (prior) start-of-
form
            start = LCtext.indexOf(_p_rs_db.SF, began);      //  its start-of-form
endpoint
        }

    //  look for [<table...>table specs</table...] recursively: load and replace specs

        String TE = "</table>";

        ended = LCtext.indexOf(TE);
        int tableNumber = 0;
        while (ended > -1)                                   //  the first end-of-table
        {
            began = LCtext.lastIndexOf("<table",ended);      //  its (prior) start-of-
table
            if (began > -1)
            {
                ended += TE.length();                        //  inclusive terminus
                tableNumber++;

            //  dbg("! - new table ("+tableNumber+"): range = ("+began+",
"+ended+")");

                _p_tb_db newTable = new _p_tb_db
                (   newText.substring(began,ended)
                ,   new Integer(tableNumber)
                ,   ez_HTML
                );
                ez_HTML.getTableDB().addTable(newTable);

                StringBuffer text = new StringBuffer();
                text.append(newText.substring(0,began))      //  preceeding text
                    .append(newTable.addCaption(true))       //  superior caption, if
any
                    .append(newTable.addCaption(false))      //  inferior caption, if
any, in superior position
                    .append(_p_rs_db.PF)
                    .append(_p_rs_db.TB)                     //  private table tag
                    .append(_p_rs_db.BL)
                    .append(_p_rs_db.VL)
                    .append(_p_rs_db.EQ)
                    .append(newTable.theTableNumber)
                    .append(_p_rs_db.SF)                     //  substitute text, e.g.
<tblt value=1>
            //      .append(newTable.addCaption(false))      //  inferior caption, if
any; shifted to superior position
                    .append(newText.substring(ended));       //  succeeding text

                newText = text.toString();
                LCtext = newText.toLowerCase();
                ended = LCtext.indexOf(TE);
            }
            else
            {
                ended = -1;                                     //  format error exit path
            }
        }

    //  look for [<body...>body text</body...]

        began = LCtext.indexOf("<body");
        if (began > -1)
        {
            int body = LCtext.indexOf(">",began)+1;      //  start of body
            if (body > 1 && body < LCtext.length())
            {
            //  parse body text attributes

                String tagText = LCtext.substring(began+1,body-1);

                ez_HTML.setLinkColor( _o_tg_db.parseColorValue(
_o_tg_db.getTagAttribString( tagText, "link"   ), 0x0000ff));
                ez_HTML.setForeColor( _o_tg_db.parseColorValue(
_o_tg_db.getTagAttribString( tagText, "text"   ), ez_HTML.getForeColor().getRGB()));
                ez_HTML.setBackColor( _o_tg_db.parseColorValue(
_o_tg_db.getTagAttribString( tagText, "bgcolor"), ez_HTML.getBackColor().getRGB()));

                String fileName = _o_tg_db.getTagAttribString(tagText,"background");

                if (fileName != null)
                {
                    ez_HTML.setBackImage(ez_HTML.getImageDB().fetchImage(fileName));
                }

                ez_HTML.setForeground(ez_HTML.getForeColor());
                ez_HTML.setBackground(ez_HTML.getBackColor());

                ended = LCtext.indexOf("</body",body);
                if (ended > -1)
                {
                    return newText.substring(body,ended);   //  nice formatting
                }
            }
        }
        return newText;
    }

    public static String lineCanon(String itsText, boolean isHTML)
    {
        if (itsText == null)
        ||  itsText.length() == 0)  return itsText;

        String newSeparator = isHTML? BL: PR;               //  target return code
        String searchSeparators[] =                         //  replacable return
codes
            {   "\n"+"\r"                                   //  chr(10), chr(13)
            ,   "\r"+"\n"                                   //  chr(13), chr(10)

            ,   "\r"                                        //  chr(13)
            ,   "\n"                                        //  chr(10)
            ,   "\f"                                        //  chr(12)
        //  ,   "\t"                                        //  chr(??)
            };
        String newText = new String(itsText);

        for (int i = 0; i < searchSeparators.length; i++)   //  replace old with new
        {
            String oldSeparator = new String(searchSeparators[i]);
            newText = replaceString(newText,oldSeparator,newSeparator);
        }

        int oldLen = newText.length(), newLen = 0;
        if (isHTML) do                                          //  prune double spaces &&
paragraph codes
        {
            oldLen = newText.length();
            newText = replaceString(newText,BL+BL,BL);
            newText = replaceString(newText,PR+PR,PR);
            newLen = newText.length();
        }
        while (oldLen > newLen);
        else
            newText = replaceString(newText,"\t","        ");

        return newText;
    }

    private static String replaceString
    (   String theString                                        //  original string
    ,   String oldString                                        //  replace-ee
    ,   String newString                                        //  replace-or
    )
    {
        char[] tag = new char[1];
        String piece = MT, theTag = MT;
        StringBuffer out = new StringBuffer();

        int oldIndex = -1, oldEnd = 0, n = 0;
        int oldLength = oldString.length();

        boolean newWhite = newString.equals(BL);

        while ((oldIndex = theString.indexOf(oldString, oldEnd)) > -1)
        {
            piece = theString.substring(oldEnd,oldIndex);   //  got it? get it!
good...
            out.append(piece);                              //  add the piece before
the next old string

        //  don't convert returns into white space after tags (browser fashion)

            if (newWhite && (n = out.length()) > 0)
            {
                tag[0] = out.charAt(n-1);
                theTag = new String(tag);
            }

            if (!(newWhite && theTag.toLowerCase().equals(">")))
                out.append(newString);                      //  get the replacement

            oldEnd = oldIndex + oldLength;                  //  skip to the end of
this old string
        }                                                   //  nothing more to
replace
        if (oldEnd < theString.length())
        {
            piece = theString.substring(oldEnd);
            out.append(piece);                              //  add the piece after
the last old string
        }
        return out.toString();
    }
}

/**
 *  %W% %E% Everett Stoub
 *
 *  Copyright (c) '96-'97 ION Web Products, Inc. All Rights Reserved.
 *
 *  object with static methods to parse html tags
 *
 *  this class is not downloaded unless needed
 *
 *  @author Everett Stoub %I%, %G%
 */
import java.awt.*;

public class _o_tg_db extends Object
{
    public static final int getTagAlignment
    (   String tagText
    ,   int itsDefault
    )
    {
        return getTagAlignment(tagText, _p_rs_db.AL, itsDefault);
    }

    public static final int getTagAlignment
    (   String tagText
    ,   String attrib
    ,   int itsDefault
    )
    {
        String itsAlignment = getTagAttribString(tagText, attrib);
        if (itsAlignment != null)
        {
            itsAlignment = itsAlignment.toLowerCase();
            if (itsAlignment.equals("left"))    return _p_rs_db.LEFT;
            if (itsAlignment.equals("center"))  return _p_rs_db.CENTER;
            if (itsAlignment.equals("right"))   return _p_rs_db.RIGHT;
            if (itsAlignment.equals("top"))     return _p_rs_db.TOP;
            if (itsAlignment.equals("middle"))  return _p_rs_db.MIDDLE;
            if (itsAlignment.equals("bottom"))  return _p_rs_db.BOTTOM;
        }
        return itsDefault;
    }

    public static final int getTagAttribInt
    (   String tagText
    ,   String attrib
    ,   int itsDefault
    )
    {
        String value = getTagAttribString(tagText, attrib);
        if (value == null)
            return itsDefault;
        else
            return parseInt(value, itsDefault);
    }

    public static final String getTagAttribString
    (   String itsText
    ,   String attrib
    )
    {
        String tagText = itsText.toLowerCase();
        int i = tagText.indexOf(attrib), L = attrib.length();   //  look for source
attribute
        if (i > -1)
        {
            int a = tagText.indexOf(_p_rs_db.EQ, i + L) + 1;    //  look for "="
assignment command
            if (0 < a && a < i + L + 2)                         //  avoid assignment
confusion
            {
                String search = "\"\' \t\n\r>";                 //  quotes position <
2
                int j = a, m = 0;                               //  start after "="
character
                int n = tagText.length();
                while (j < n                                    //  skip single quote
or whitespace or close tag
```

```
                && (m = search.indexOf(tagText.charAt(j))) > 0) j++;
                if (j < n)
                {
                    if (m == 0)                                    //  double quote:
match for balance
                    {
                        m = tagText.indexOf(search.charAt(m),j+1);
                        if (m > -1) return itsText.substring(j+1,m);
                    }

                //  attribute is not enclosed in double quotes: use whitespace to
delimit

                    int k = j+1;
                    while (k < n                                   //  skip until any
quote or whitespace or close tag
                    && search.indexOf(tagText.charAt(k)) == -1) k++;

                    return itsText.substring(j,k).trim();
                }
            }
        }
        return null;
    }

    public static final Color getTagColor
    (   String tagText  //  lower case
    ,   int defaultRGB
    )
    {
        return parseColorValue(getTagAttribString(tagText, "color"), defaultRGB);
    }

    public static final int getTagFontIndex
    (   String tagText
    ,   int defaultIndex
    )
    {
        String sizeString = getTagAttribString(tagText, _p_rs_db.SZ);
        if (sizeString == null)
            return defaultIndex;

        int itsSizeIndex = defaultIndex;

        if (sizeString.startsWith("+"))
            try {itsSizeIndex = defaultIndex +
Integer.parseInt(sizeString.substring(1));}
            catch (NumberFormatException nf) {}
        else
        if (sizeString.startsWith("-"))
            try {itsSizeIndex = defaultIndex -
Integer.parseInt(sizeString.substring(1));}
            catch (NumberFormatException nf) {}
        else
            try {itsSizeIndex = Integer.parseInt(sizeString);}
            catch (NumberFormatException nf) {}

        return Math.max(Math.min(itsSizeIndex,16),0);    //  max index value for
readerFontSize array
    }

    public static final int getTagPerCentWidth(String tagText)
    {
        String width = getTagAttribString(tagText, "width");
        if (width != null) try
        {
            if (width.endsWith("%"))
                return -Integer.parseInt(width.substring(0,width.length()-1));
            else
                return Integer.parseInt(width);
        }
        catch (NumberFormatException nf)
        {
        }
        return 0;
    }

    public static final int getTagWidth
    (   String tagText
    ,   int defaultWidth
    )
    {
        int itsWidth = defaultWidth;
        String width = getTagAttribString(tagText, "width");
        if (width != null) try
        {
            if (width.endsWith("%"))
                itsWidth = Integer.parseInt(width.substring(0,width.length()-
1))*defaultWidth/100;
            else
                itsWidth = Integer.parseInt(width);

            itsWidth = Math.min(itsWidth,defaultWidth);
        }
        catch (NumberFormatException nf) {}
        return itsWidth;
    }

    public static final Color parseColorValue
    (   String colorValue
    ,   int defaultRGB
    )
    {
        int rgb = defaultRGB;
        if (colorValue != null)
        {
            if (colorValue.startsWith("#"))
            {
                try {rgb = Integer.parseInt(colorValue.substring(1), 16);} catch
(NumberFormatException nf) {}
            }
            else    //  try to interpret the named color
            {
                if (colorValue.equals("black"))      rgb = 0x000000; else
                if (colorValue.equals("red"))        rgb = 0xff0000; else
                if (colorValue.equals("green"))      rgb = 0x00ff00; else
                if (colorValue.equals("blue"))       rgb = 0x0000ff; else
                if (colorValue.equals("yellow"))     rgb = 0xffff00; else
                if (colorValue.equals("magenta"))    rgb = 0xff00ff; else
                if (colorValue.equals("cyan"))       rgb = 0x00ffff; else
                if (colorValue.equals("white"))      rgb = 0xffffff; else
                if (colorValue.equals("lightGray"))  rgb = 0xc0c0c0; else
                if (colorValue.equals("gray"))       rgb = 0x808080; else
                if (colorValue.equals("darkGray"))   rgb = 0x404040; else
                if (colorValue.equals("pink"))       rgb = 0xffafaf; else
                if (colorValue.equals("orange"))     rgb = 0xffc800; else

            //  maybe a hex number after all (w/o preceeding # sign)

                try {rgb = Integer.parseInt(colorValue, 16);} catch
(NumberFormatException nf) {}
            }
        }
        return new Color(rgb);
    }

    public static final int parseInt
    (   String value
    ,   int itsDefault
    )
    {
        try
        {
            return Integer.parseInt(value);
        }
        catch (NumberFormatException nf)
        {
            return itsDefault;
        }
    }
}

/**
 *  %W% %E% Everett Stoub
 *
 *  Copyright (c) '96-'97 ION Web Products, Inc. All Rights Reserved.
 *
 *  a pagable panel to display multicolumn text and graphics
 *
 *  HTML tag parsing honors the following commands and attributes
 *
 *    • <a>...</a>                  anchor tag
 *          href                        relative or absolute url for hyperlink
 *
 *    • <address>...</address>      address (new italic paragraph)
 *
 *    • <applet>...</applet>        applet tag & range: range is hidden, applet is
not launched
 *
 *    • <b>...</b>                  bold
 *
 *    • <basefont>...</basefont>    basic font size for document; </basefont>
reverts to 4
 *          size=value                  relative or absolute index size adjustment
 *
 *    • <big>...</big>              increase font size one step
 *
 *    • <blockquote>...</blockquote>    blockquote (left & right margin indentation)
 *
 *    • <br>...                     new paragraph, no extra halfspace
 *          align=type                  left, center, right
 *
 *    • <center>...</center>        center justify
 *
 *    • <cite>...</cite>            citation (italic)
 *
 *    • <code>...</code>            code (monospaced)
 *
 *    • <dir>...</dir>              unordered directory list
 *          type=bullet type            circle, square, or disc
 *
 *    • <dd>...</dd>                definition list definition component
 *
 *    • <dl>...</dl>                unordered (non-bulleted) definition list
 *
 *    • <dt>...</dt>                definition list term component
 *
 *    • <em>...</em>                emphasis (italic)
 *
 *    • <font>...</font>            font color and size; </font> reverts to normal
 *          color=color                 rgb or color name
 *          size=value                  size index value
 *
 *    • <form>...</form>            form specification domain
 *          action=url                  server contact for response
 *          enctype=encoding            element value coding system
 *          method=style                either get or post
 *
 *    • <hr>                        horizontal r 'e
 *          align=type                  left [illegible] r, right
 *          size=pixels                     vertical rule size
 *          noshade                         sets flat rounded rect rule
 *          width=value or %                specify width up to column width value
 *
 *      • <h1...6>...</h1...6>,         heading levels 1 thru 6
 *          align=type                      left, center, right
 *
 *      • <i>...</i>                    italic
 *
 *      • <img>                         insert .gif or .jpeg image
 *          align=type                      bottom, middle, or top
 *          src=url                         image url
 *
 *      • <input>                       form input element specification
 *          align=type                      top, middle, or bottom (image)
 *          border=n                        pixel border thickness (image)
 *          checked                         marks radio or checkbox initially selected
 *          maxlength=n                     max # of chars to accept (file or
password)
 *          name=name                       passed to designated url
 *          size=n                          max # of chars to display (file or
password)
 *          src=url                         image url
 *          type=type                       button, checkbox, file, hidden, image,
password, radio, reset, submit, text
 *          value=string                    passed to designated url
 *
 *      • <kbd>...</kbd>                keyboard (monospaced)
 *
 *      • <li>...                       list item
 *          type=bullet type                for unordered: circle, square, or disc
 *          type=number type                for ordered: A, a, I, i, or 1
 *          start=int                       overriding ordered list number
 *
 *      • <menu>...</menu>              unordered menu list
 *          type=bullet type                circle, square, or disc
 *
 *      • <ol>...</ol>                  ordered (nested) list
 *          type=number type                A, a, I, i, or 1
 *          start=int                       overriding ordered list number
 *
 *      • <option>...</option>          pop-up input list element definition
 *          selected                        initially select element
 *          value=string                    passed to designated url (instead of
<optiion> contents)
 *
 *      • <p>...</p>                    new paragraph, extra halfspace
 *          align=type                      left, center, right
 *
 *      • <plaintext>                   plaintext (rendered literally, with monospaced
font): no terminal
 *
 *      • <pre>...</pre>                pre-formatted text (rendered literally, with
monospaced font)
 *
 *      • <select>...</select>          pop-up input list specification domain
 *          multiple                        allow multiple selections
 *          name=name                       passed to designated url
 *          size=n                          number of items to display
 *
 *      • <small>...</small>            decrease font size one step
 *
 *      • <strike>...</strike>          strikethrough text
 *
 *      • <strong>...</strong>          strong (bold)
 *
 *      • <sub>...</sub>                subscript
 *
 *      • <sup>...</sup>                superscript
 *
 *    • <table>...</table>            table tag & range (replaced with hot image)
 *
 *    • <textarea>...</textarea>      text input area specification domain
 *          cols=n                        number of characters to display in a row
 *          name=name                     passed to designated url
 *          rows=n                        number of lines to display in area
 *
 *    • <tt>...</tt>                  teletype (monospaced)
 *
 *    • <u>...</u>                    underline
 *
 *    • <ul>...</ul>                  unordered (nested) list
 *          type=bullet type              circle, square, or disc
 *
 *    • <var>...</var>                variable (italic)
 *
 *    • <!--...-->                    html comments (stripped out)
 *
 *  @author Everett Stoub %I%, %G%
 */
import java.awt.*;
import java.awt.image.*;
import java.applet.*;
import java.net.*;
import java.io.*;
import java.util.*;

public final class _p_rs_db extends Panel
{
    public final static int     LEFT        =    0;    //  position
    public final static int     CENTER      =    1;    //  position
    public final static int     RIGHT       =    2;    //  position
    public final static int     TOP         =    3;    //  position
    public final static int     MIDDLE      =    4;    //  position
    public final static int     BOTTOM      =    5;    //  position
    public final static int     POSITION    =    7;    //  position mask
    public final static int     UNDERLINE   =    8;    //  text
    public final static int     STRIKETHRU  =   16;    //  text
    public final static int     SUBSCRIPT   =   32;    //  text
    public final static int     SUPSCRIPT   =   64;    //  text
    public final static int     HYPERLINK   =  128;    //  text
    public final static int     LEFTINDENT  =  256;    //  text
    public final static int     RIGHTINDENT =  512;    //  text
    public final static int     HIDDENTEXT  = 1024;    //  text
    public final static int     UNORDERLIST = 2048;    //  text
    public final static int     ORDEREDLIST = 4096;    //  text

    public final static String  AD          = "address";
    public final static String  AL          = "align";
    public final static String  AN          = "a";
    public final static String  AP          = "applet";
    public final static String  BD          = "b";
    public final static String  BF          = "basefont";
    public final static String  BG          = "big";
    public final static String  BL          = " ";
    public final static String  BQ          = "blockquote";
    public final static String  BR          = "br";
    public final static String  CD          = "code";
    public final static String  CJ          = "center";
    public final static String  CK          = "checked";
    public final static String  CR          = "Courier";
    public final static String  CT          = "cite";
    public final static String  DD          = "dd";
    public final static String  DL          = "dl";
    public final static String  DT          = "dt";
    public final static String  DR          = "dir";
    public final static String  EM          = "em";
    public final static String  EN          = "elementNumber";  //  private form
element attribute
    public final static String  EQ          = "=";
    public final static String  FM          = "frm";          //  private form tage
    public final static String  FN          = "formNumber";   //  private form attribute
    public final static String  FS          = "TimesRoman";
    public final static String  FT          = "font";
    public final static String  HF          = "href";
    public final static String  HN          = "h";
    public final static String  HR          = "hr";
    public final static String  IM          = "img";
    public final static String  IN          = "input";
    public final static String  IT          = "i";
    public final static String  KB          = "kbd";
    public final static String  LI          = "li";
    public final static String  ML          = "menu";
    public final static String  MT          = "";
    public final static String  MX          = "maxlength";
    public final static String  NM          = "name";
    public final static String  OL          = "ol";
    public final static String  OP          = "option";
    public final static String  PA          = "p";
    public final static String  PF          = "<";
    public final static String  PR          = "pre";
    public final static String  PT          = "plaintext";
    public final static String  SB          = "sub";
    public final static String  SF          = ">";
    public final static String  SG          = "strong";
    public final static String  SK          = "strike";
    public final static String  SL          = "/";
    public final static String  SM          = "small";
    public final static String  SP          = "sup";
    public final static String  SR          = "src";
    public final static String  ST          = "select";
    public final static String  SZ          = "size";
    public final static String  TA          = "textarea";
    public final static String  TB          = "tblt";         //  private table tag
    public final static String  TP          = "type";
    public final static String  TT          = "tt";
    public final static String  UL          = "ul";
    public final static String  UN          = "u";
    public final static String  VA          = "var";
    public final static String  VL          = "value";
    public final static String  ZI          = "Release for fullsize image view...";
    public final static String  ZM          = "Click to zoom image...";
    public final static String  ZN          = "Release for new reader screen...";
    public final static String  ZW          = "Wait for new reader screen...";
    public final static String  ZX          = "Wait for fullsize image view...";
    public final static String  ZZ          = ".. ... ... ... ... ... ... ...";

    public ez_html          ez_HTML             = null;
    private _f_rs_db        frame               = null;
    private _o_nt_db        charEntConverter    = null;

    public String           readerContent       = null;
    public Insets           insets              = new Insets(0,0,0,0);
    private int             maxColumns          = 15;
    private Dimension       minimumSize         = new Dimension( 0, 0);
    private Dimension       preferredSize       = new Dimension(-1,-1);
    private Image           readerScreen        = null;
    public boolean          zoomable            = false;

    public int              readerFontSize[]    = { 8, 9,10,11,12,14,18,24,30,36,42,48,56,64,72,80,96};
    public int              readerFootIndex     = 2;      //  this 2-step decrement is also hard coded in stepFontSize()
    public int              readerSizeIndex     = 4;      //  default for on-webPage article rendering
    public int              readerEnWidth       = 6;
    public Font             readerScreenFont    = null;
    public Font             readerFooterFont    = null;
    public int              footerTextHeight    = 12;

    public int          readerNumCols       = 1;
    public int          readerLeading       = 0;
    public int          readerPageNum       = 1;
    public int          readerNumPages      = 0;
    public int          readerEstPages      = 1;
    public int          readerEndIndex      = 1;
    public int          readerStartIndex    = 0;
    public int          readerTextLength    = 0;
    public boolean      readerScreenDirty   = true;            //  true if panel
needs redraw
    public boolean      readerFrame         = false;           //  true if panel
owned by frame
    public boolean      readerHeader        = false;           //  true if article
begins with `title
    public boolean      readerMarker        = false;           //  true to present
markers
    public boolean      readerFooter        = false;           //  true to present
footer

    public Vector       readerPageImage     = new Vector(50);
    public Vector       readerListAccts     = new Vector(50);
    public Vector       readerLinkHeads     = new Vector(50);
    public Vector       readerLinkRects     = new Vector(50);
    public Vector       readerZoomHeads     = new Vector(50);
    public Vector       readerZoomRects     = new Vector(50);
    public Vector       readerPageComps     = new Vector(50);
    public Vector       readerPageStart     = new Vector(50);
    public Vector       readerFontStart     = new Vector(50);
    public Vector       readerLinkStart     = new Vector(50);
    public Vector       readerTintStart     = new Vector(50);
    public Vector       readerTypeStart     = new Vector(50);

    public Rectangle    readerRect          = new Rectangle();

//  variables principally for paintArea

    public int          border              =  0;
    public int          padding             =  0;
    public int          spacing             =  0;

    public int          maxWidth            =  0;
    public int          maxHeight           =  0;
    public int          controlSize         =  8;

    public boolean      firstWord           = true;            //  current setting
    public boolean      lastColumn          = false;           //  current setting
    public boolean      lastRow             = false;           //  current setting
    public boolean      termDefinition      = false;
    public boolean      plaintext           = false;           //  true blocks tag
parsing

    public Rectangle    aheadRect           = new Rectangle();

    public boolean      newListItem         = false;           //  current page
pending item
    public Vector       listAccount         = new Vector();    //  current page
(nested) list elements
    public Vector       linkHeads           = new Vector();    //  current page
    public Vector       linkRects           = new Vector();    //  current page
    public Vector       zoomHeads           = new Vector();    //  current page
    public Vector       zoomRects           = new Vector();    //  current page
    public Vector       pageComps           = new Vector();    //  current page

    public Font         textFont            = null;            //  current setting
    public FontMetrics  textMetrics         = null;            //  current setting
    public Color        textColor           = null;            //  current setting
    public int          textHeight          = 0;               //  current setting
    public int          textHalfSpace       = 0;               //  current setting
    public int          textAlign           = LEFT;            //  current setting

    public int          colNum = 0, colWidth = 0, colSpacing = 0, numColumns = 0,
tagLength = 0;
    public int          endIndex = 0, nextWidth = 0, xb = 0, te = 0, xe = 0, yb = 0,
ye = 0;

    public boolean      markings            = false;            //  current word

    public boolean      underlining         = false;            //  current word
    public boolean      strikethrough       = false;            //  current word
    public boolean      subscript           = false;            //  current word
    public boolean      supscript           = false;            //  current word
    public boolean      hyperLink           = false;            //  current word
    public boolean      leftIndent          = false;            //  current word
    public boolean      rightIndent         = false;            //  current word
    public boolean      hidingText          = false;            //  current word
    public boolean      unorderList         = false;            //  current word
    public boolean      orderedList         = false;            //  current word
    public URL          hyperHead           = null;             //  current word

    public Vector       lineImages          = new Vector();     //  current row
    public Vector       dataValues          = new Vector();     //  current row
    public Vector       wordBlocks          = new Vector();     //  current row
    public Vector       hyperLinks          = new Vector();     //  current row
    public Vector       styleCodes          = new Vector();     //  current row
    public Vector       wordColors          = new Vector();     //  current row
    public Vector       charsFonts          = new Vector();     //  current row

    public final boolean addBlock(String tagBlock, Graphics g, boolean count)
    {
    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").addBlock("+tagBloc
k+")");

        if (tagBlock == null)
            return true;

        StringTokenizer words =
            new StringTokenizer(tagBlock,BL,true);      //  "true" returns blanks (for
counting)
        String nextWord = MT;

        while (words.hasMoreTokens())
        {
            nextWord = words.nextToken();
            nextWidth = addWord(nextWord, false, count);

            if (te + nextWidth < xe - indent(true))     //  it will fit on this line,
so add it on
            {
                te += nextWidth;
            }
            else                                        //  it won't fit if added, so
draw the line
            {
                endIndex += paintRow(g, 1);             //  try to punch out prior
pending line elements
                te = xb + indent(false);
                if (yb > ye)                            //  start a new column?
                {
                    if (lastColumn) return false;       //  no more columns left to
fill
                    nextColumn();
                }
                if (nextWord.equals(BL))                //  drop leading blank at new
line
                {
                    endIndex++;                         //  count the blank
                    clearRowElements();
                    savePageData
                    (   readerPageNum
                    ,   endIndex
                    );                              //  for the next page

                    nextWord = MT;
                    nextWidth = MT.length();
                }
                te += nextWidth;                    //  start a new line with,
possibly, a monster!

                if (te > xe - indent(true) && preferredSize.width > 0)
                {
                    preferredSize.width = Math.max
                    (   insets.left + insets.right + 2
                    +   nextWidth
                    ,   preferredSize.width
                    );

                //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").addBLock(); new
preferredSize("+preferredSize.width+","+preferredSize.height+") = resize() +
("+(size().width - preferredSize.width)+","+(size().height - preferredSize.height)+"):
nextWord = ["+nextWord+"]");
                }
                while (te > xe - indent(true))      //  a monster: shorten it by
breaking the word
                {
                    te = xb + indent(false);        //  start the new line again
                    String monster = nextWord;      //  this would be a great
place for a hyphenator
                    nextWord = MT;
                    int e = monster.length();
                    while (te + nextWidth > xe - indent(true) && e > 0)
                    {                               //  transfer letters back
until fit (one line)
                        clearRowElements();
                        nextWord = monster.substring(e-1,e) + nextWord;
                        monster = monster.substring(0,e-1);
                    //  dbg("... monster = ["+monster+"], nextWord = ["+nextWord+"]");
                        e = monster.length();
                        nextWidth = addWord(monster, true, count);
                    }
                    te += nextWidth;
                    endIndex += paintRow(g, 0);     //  try to punch out pending
line elements
                    te = xb + indent(false);        //  start the new line again
                    if (yb > ye)                    //  start a new column?
                    {
                        if (lastColumn) return false;   //  no more columns left to
fill
                        nextColumn();
                    }
                    nextWidth = addWord(nextWord, false, count);
                    te += nextWidth;
                }
            }
        }
        return true;                                //  no more words in this tag
block
    }

    private final int addTag(String itsText, Graphics g)
    {
    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").addTag("+itsText+"
)");

```

```
        String tagText = itsText.toLowerCase();
        int addIndex = 0;
        tagLength = tagText.length() + 2;

        if (tagText.equals(PA)                              //  an html new paragraph
tag w/o attributes
        ||  tagText.startsWith(PA+BL)                       //  an html new paragraph
tag w/  attributes
        ||  tagText.startsWith(BR)                          //  an html new line tag
        ||  tagText.equals(AD))                             //  an html address tag
        {
            addIndex += paintRow(g,0);                      //  try to punch out
pending line elements
            if (!startNewParagraph(tagText.startsWith(BR)?0:1)) return addIndex;
            textAlign =
_o_tg_db.getTagAlignment(tagText,(tagText.startsWith(PA)?LEFT:textAlign));
            if (tagText.equals(AD)) this.setFont(new
Font(textFont.getName(),textFont.getStyle() + Font.ITALIC,textFont.getSize()));
        }
        else if (tagText.equals(SL+PA)                      //  an html end paragraph
tag
        ||      tagText.equals(SL+AD))                      //  an html end address
tag
        {
            addIndex += paintRow(g,0);                      //  try to punch out
pending line elements
            textAlign = LEFT;                               //  default alignment
            if (tagText.equals(SL+AD)) this.setFont(new
Font(textFont.getName(),textFont.getStyle() - Font.ITALIC,textFont.getSize()));
        }
        else if (tagText.startsWith(IM))                    //  an html image tag
        {
            int newAddIndex = _o_im_db.placeImage(tagText, ez_HTML, this, ++addIndex,
g); //  bias for sign detect
            if (newAddIndex < 0) return ++newAddIndex; else addIndex = --newAddIndex;
        //  reverse bias
        }
        else if (tagText.startsWith(HR))                    //  an html horizontal
rule tag
        {
            int newAddIndex = _o_hr_db.placeRule(tagText, this, ++addIndex, g);
        //  bias for sign detect
            if (newAddIndex < 0) return ++newAddIndex; else addIndex = --newAddIndex;
        //  reverse bias
        }
        else if (tagText.startsWith(TB))                    //  a preprocessed html
table tag
        {
            int newAddIndex = _o_tb_db.placeTable(tagText, ez_HTML, this, ++addIndex,
g); //  bias for sign detect
            if (newAddIndex < 0) return ++newAddIndex; else addIndex = --newAddIndex;
        //  reverse bias
        }
        else if (tagText.startsWith(HN)                     //  an html heading level
tag
             && (tagText.length() < 3                       //  either without
attributes
             ||  tagText.substring(2,3).equals(BL)))        //  or with attributes to
follow
        {
            addIndex += paintRow(g,0);                      //  try to punch out
pending line elements
            if (!startNewParagraph(1)) return addIndex;
            textAlign = _o_tg_db.getTagAlignment(tagText, LEFT);
            this.setFont(new
Font(textFont.getName(),Font.BOLD,readerFontSize[readerSizeIndex + 4 -
_o_tg_db.parseInt(tagText.substring(1,2), 4)]));
        }
        else if (tagText.startsWith(AN+BL))                 //  an html anchor tag
        {
            try
            {
                hyperHead = new URL(ez_HTML.getDocumentBase(),
_o_tg_db.getTagAttribString(tagText,HF));
                hyperLink = true;
                textColor = ez_HTML.linkColor;
            }
            catch (MalformedURLException m) {};
        }
        else if (tagText.equals(SL+AN))                        //  an html end anchor tag
        {
            if (hyperLink)
            {
                hyperHead = null;
                hyperLink = false;
                textColor = getForeground();
            }
        }
        else if (tagText.equals(BD)                            //  an html bold tag
             ||  tagText.equals(SG))                           //  an html strong tag
        {
            this.setFont(new Font(textFont.getName(),textFont.getStyle() +
Font.BOLD,textFont.getSize()));
        }
        else if (tagText.equals(SL+BD)                         //  an html end bold tag
             ||  tagText.equals(SL+SG))                        //  an html end strong tag
        {
            this.setFont(new Font(textFont.getName(),textFont.getStyle() -
Font.BOLD,textFont.getSize()));
        }
        else if (tagText.equals(IT)                            //  an html italic tag
             ||  tagText.equals(CT)                            //  an html citation tag
             ||  tagText.equals(EM)                            //  an html emphasis tag
             ||  tagText.equals(VA))                           //  an html variable tag
        {
            this.setFont(new Font(textFont.getName(),textFont.getStyle() +
Font.ITALIC,textFont.getSize()));
        }
        else if (tagText.equals(SL+IT)                         //  an html end italic tag
             ||  tagText.equals(SL+CT)                         //  an html end citation
tag
             ||  tagText.equals(SL+EM)                         //  an html end emphasis
tag
             ||  tagText.equals(SL+VA))                        //  an html end variable
tag
        {
            this.setFont(new Font(textFont.getName(),textFont.getStyle() -
Font.ITALIC,textFont.getSize()));
        }
        else if (tagText.startsWith(FT))                       //  an html font tag
        {
            this.setFont(new
Font(textFont.getName(),textFont.getStyle(),readerFontSize[_o_tg_db.getTagFontIndex(ta
gText,readerSizeIndex)]),_o_tg_db.getTagColor(tagText,textColor.getRGB()));
        }
        else if (tagText.equals(SL+FT)                         //  an html end font tag
             || (tagText.startsWith(SL+HN)                     //  an html end heading
tag
             &&  tagText.length() < 4))
        {
            this.setFont(readerScreenFont,getForeground());
            if (tagText.startsWith(SL+HN)                      //  an html end heading
level tag
            &&  tagText.length() < 4)
            {
                addIndex += paintRow(g,0);                     //  try to punch out
pending line elements
                if (!startNewParagraph(0)) return addIndex;
            }
        }
        else if (tagText.startsWith(LI)                     //  an html list item tag
             ||  tagText.equals(DT)                         //  an html definition
term tag
             ||  tagText.equals(DD))                        //  an html definition
component tag
        {
            int newAddIndex = _o_li_db.addItem(tagText, itsText, this, ++addIndex, g);
        //  bias for sign detect
            if (newAddIndex < 0) return ++newAddIndex; else addIndex = --newAddIndex;
        //  reverse bias
        }
        else if (tagText.startsWith(UL)                     //  an html unordered list
tag
             ||  tagText.startsWith(OL)                     //  an html ordered list
tag
             ||  tagText.startsWith(ML)                     //  an html menu list tag
             ||  tagText.equals(DL)                         //  an html definition
list tag
             ||  tagText.startsWith(DR))                    //  an html directory list
tag
        {
            int newAddIndex = _o_li_db.addList(tagText, itsText, this, ++addIndex, g);
        //  bias for sign detect
            if (newAddIndex < 0) return ++newAddIndex; else addIndex = --newAddIndex;
        //  reverse bias
        }
        else if (tagText.equals(SL+UL)                      //  an html end unordered
list tag
             ||  tagText.equals(SL+OL)                      //  an html end ordered
list tag
             ||  tagText.equals(SL+ML)                      //  an html end menu list
tag
             ||  tagText.equals(SL+DL)                      //  an html end definition
list tag
             ||  tagText.equals(SL+DR))                     //  an html end directory
list tag
        {
            int newAddIndex = _o_li_db.endList(tagText, itsText, this, ++addIndex, g);
        //  bias for sign detect
            if (newAddIndex < 0) return ++newAddIndex; else addIndex = --newAddIndex;
        //  reverse bias
        }
        else if (tagText.startsWith(IN))
        {
            int newAddIndex = _o_fm_db.addInput(tagText, itsText, this, ++addIndex,
g);     //  bias for sign detect
            if (newAddIndex < 0) return ++newAddIndex; else addIndex = --newAddIndex;
        //  reverse bias
        }
        else if (tagText.startsWith(OP))
        {
            int newAddIndex = _o_fm_db.addOption(tagText, itsText, this, ++addIndex,
g); //  bias for sign detect
            if (newAddIndex < 0) return ++newAddIndex; else addIndex = --newAddIndex;
        //  reverse bias
        }
        else if (tagText.equals(SL+OP))
        {
            int newAddIndex = _o_fm_db.endOption(tagText, itsText, this, ++addIndex,
g); //  bias for sign detect
            if (newAddIndex < 0) return ++newAddIndex; else addIndex = --newAddIndex;
        //  reverse bias
        }
        else if (tagText.startsWith(ST))
        {
            int newAddIndex = _o_fm_db.addSelector(tagText, itsText, this, ++addIndex,
g); //  bias for sign detect
            if (newAddIndex < 0) return ++newAddIndex; else addIndex = --newAddIndex;
        //  reverse bias
        }
        else if (tagText.equals(SL+ST))
        {
            int newAddIndex = _o_fm_db.endSelector(tagText, itsText, this, ++addIndex,
g); // bias for sign detect
            if (newAddIndex < 0) return ++newAddIndex; else addIndex = --newAddIndex;
        // reverse bias
        }
        else if (tagText.startsWith(TA))
        {
            int newAddIndex = _o_fm_db.addTextArea(tagText, itsText, this, ++addIndex,
g); // bias for sign detect
            if (newAddIndex < 0) return ++newAddIndex; else addIndex = --newAddIndex;
        // reverse bias
        }
        else if (tagText.equals(SL+TA))
        {
            int newAddIndex = _o_fm_db.endTextArea(tagText, itsText, this, ++addIndex,
g); // bias for sign detect
            if (newAddIndex < 0) return ++newAddIndex; else addIndex = --newAddIndex;
        // reverse bias
        }
        else if (tagText.startsWith(FM))
        {
            int newAddIndex = _o_fm_db.addForm(tagText, itsText, this, ++addIndex, g);
        // bias for sign detect
            if (newAddIndex < 0) return ++newAddIndex; else addIndex = --newAddIndex;
        // reverse bias
        }
        else if (tagText.equals(SL+FM))
        {
            int newAddIndex = _o_fm_db.endForm(tagText, itsText, this, ++addIndex, g);
        // bias for sign detect
            if (newAddIndex < 0) return ++newAddIndex; else addIndex = --newAddIndex;
        // reverse bias
        }
        else if (tagText.equals(BG)                         // an html big tag
             ||  tagText.equals(SL+SM)                      // an html end small tag
             ||  tagText.equals(SM)                         // an html small tag
             ||  tagText.equals(SL+BG))                     // an html end big tag
        {
            this.setFont(new
Font(textFont.getName(),textFont.getStyle(),readerFontSize[Math.max(Math.min(getFontIn
dex(textFont,readerSizeIndex)+((tagText.equals(BG) || tagText.equals(SL+SM))?(1):(-
1)),readerFontSize.length - 1),0)]));
        }
        else if (tagText.equals(BQ)                         // an html blockquote tag
             ||  tagText.equals(SL+BQ))                     // an html end blockquote
tag
        {
            leftIndent = rightIndent = tagText.equals(BQ);
        }
        else if (tagText.equals(CJ)                         // an html center tag
             ||  tagText.equals(SL+CJ))                     // an html end center tag
        {
            textAlign = tagText.equals(CJ)?CENTER:LEFT;
        }
        else if (tagText.equals(SK)                         // an html strikethrough
tag
             ||  tagText.equals(SL+SK))                     // an html end
strikethrough tag
        {
            strikethrough = tagText.equals(SK);
        }
        else if (tagText.equals(UN)                         // an html underline tag
             ||  tagText.equals(SL+UN))                     // an html end underline
tag
        {
            underlining = tagText.equals(UN);
        }
        else if (tagText.equals(SB)                         // an html subscript tag
            ||  tagText.equals(SL+SB))                  //  an html end subscript
tag
        {
            subscript = tagText.equals(SB);
        }
        else if (tagText.equals(SP)                     //  an html superscript
tag
            ||  tagText.equals(SL+SP))                  //  an html end
superscript tag
        {
            supscript = tagText.equals(SP);
        }
        else if (tagText.equals(CD)                     //  an html code tag:
monospaced font
            ||  tagText.equals(PR)                      //  an html pre tag:
monospaced font
            ||  tagText.equals(PT)                      //  an html plaintext tag:
monospaced font
            ||  tagText.equals(TT)                      //  an html teletype tag:
monospaced font
            ||  tagText.equals(KB))                     //  an html keyboard tag:
monospaced font
        {
            this.setFont(new Font(CR,Font.PLAIN,textFont.getSize()));
            if (tagText.equals(PT)) plaintext = true;   //  if true, further tag
parsing is blocked
        }
        else if (tagText.equals(SL+CD)                  //  an html end code tag:
standard font
            ||  tagText.equals(SL+PR)                   //  an html end pre tag:
standard font
            ||  tagText.equals(SL+TT)                   //  an html end teletype
tag: standard font
            ||  tagText.equals(SL+KB))                  //  an html end keyboard
tag: standard font
        {
            this.setFont(new Font(FS,Font.PLAIN,textFont.getSize()));
        }
        else if (tagText.startsWith(BF)                 //  an html basefont tag
            ||  tagText.equals(SL+BF))                  //  an html end basefont
tag
        {
            readerSizeIndex =
(tagText.startsWith(BF))?_o_tg_db.getTagFontIndex(tagText,readerSizeIndex):4;
            readerScreenFont = new Font(FS, Font.PLAIN,
readerFontSize[readerSizeIndex]);
            this.setFont(readerScreenFont);
        }
        else                                            //  whatever has not been
handled is ignored but counted
        {
            dbg("tag = <"+tagText+"> was not handled; tagLength = "+tagLength);
        }

        lineImages.addElement(null);
        dataValues.addElement
        (   new Rectangle
            (   tagLength                               //  "x"         increment
for endIndex (character count)
            ,   0                                       //  "y"         for
vertical placement of image (ascent)
            ,   0                                       //  "width"     element
width on line
            ,   0                                       //  "height"    element
drop below baseline (descent)
            )
        );
        wordBlocks.addElement(MT);
        hyperLinks.addElement(null);

        styleCodes.addElement(styleCode());
        wordColors.addElement(textColor);
        charsFonts.addElement(textFont);

        return addIndex;                                  //  the endIndex increment
precipitated by this tag
    }

    private final int addWord(String nextWord, boolean addHyphen, boolean count)
    {
    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").addWord(\""+nextWo
rd+"\")");

        if (nextWord == null) return 0;                   //  nothing to draw

        String block = charEntConverter.convertCharacterEntities(nextWord);
        if (hidingText) block = MT;                       //  cheap trip for hidden
text
        if (addHyphen) block = block + "-";               //  don't count this: it's
an added character

        int wIm = textMetrics.stringWidth(block);         //  all words are followed
by spaces
        int hIm = textHeight + readerLeading;

        if (firstWord && wIm > 0)
        {
            minimumSize = new Dimension
            (   wIm + insets.left + insets.right + 2
            ,   hIm + insets.top + insets.bottom + 2
            );
        //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").addWord(): new
minimumSize("+minimumSize.width+", "+minimumSize.height+") for ["+nextWord+"]");

            firstWord = false;
        }

        int maxAscent = (textHeight+2)*2/3 + readerLeading; //  ascent and descent are
wrong in some browsers

        lineImages.addElement(null);
        dataValues.addElement
        (   new Rectangle
            (   (count?nextWord.length():0)               //  "x"        increment
for endIndex (character count)
            ,   maxAscent                                 //  "y"        for
vertical placement of image (ascent)
            ,   wIm                                       //  "width"    element
width on line
            ,   hIm - maxAscent                           //  "height"   element
drop below baseline (descent)
            )
        );
        wordBlocks.addElement(new String(block));
        hyperLinks.addElement(hyperHead);

        styleCodes.addElement(styleCode());
        wordColors.addElement(textColor);
        charsFonts.addElement(textFont);

        return wIm;                                       //  the width of this
element on the line
    }

    public final int calculateReaderColumns()
    {
        return calculateReaderColumns(this.size().width);    //  the width of this
panel
    }

    public final int calculateReaderColumns(int breadth)
    {
    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").calculateReaderCol
umns(): breadth = "+breadth);

        int numCols = breadth / standardLineWidth();
    //  dbg("... breadth = "+breadth+" / numChar("+numChar+") /
readerEnWidth("+readerEnWidth+") = numCols("+numCols+"); maxColumns = "+maxColumns);

        if (numCols < 1)
            numCols = 1;

        return Math.min(numCols, maxColumns);
    }

    public final void clearPage()
    {
    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").clearPage()");

        newListItem = false;
        listAccount = new Vector();

        unorderList = false;
        orderedList = false;

        hyperHead = null;
        hyperLink = false;
        linkHeads = new Vector();
        linkRects = new Vector();

        zoomHeads = new Vector();
        zoomRects = new Vector();

        removeAll();
    }

    public void clearRect(int x, int y, int width, int height)
    {
    //  override, to retain transparency for background tiling if any
    }

    public final void clearRowElements()
    {
        lineImages.removeAllElements();
        dataValues.removeAllElements();
        wordBlocks.removeAllElements();
        hyperLinks.removeAllElements();
        styleCodes.removeAllElements();
        wordColors.removeAllElements();
        charsFonts.removeAllElements();
    }

    public Dimension columnSize()
    {
        return new Dimension(effectiveWidth(), readerRect.height);
    }

    private final void dbg(String s)
    {
        if (debug) System.out.println(s);
    }
    private boolean debug = true;    //  Print debugging info?

    public final void dispose()
    {
    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").dispose()");

        if (readerScreen != null)
            readerScreen.flush();
    }

    public final void doViewerSize(boolean smaller)
    {
    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").doViewerSize("+sma
ller+") # comps = "+countComponents());

        stepFontSize(smaller);

        setReaderRect();
        clearRowElements();
        resetReaderView();
        estimateNumPages();
        repaintReaderScreen();
    }

    public int effectiveWidth()
    {
        if (!markings
        ||  aheadRect.x == 0)
            return colWidth - indent(true) - indent(false);
        else
            return  Math.min
            (   aheadRect.x + indent(true)
            ,   xb + colWidth
            ) - xb - indent(true) - indent(false);
    }

    public void estimateNumPages()
    {
    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").estimateNumPages()
");

        if (readerNumPages == 0)
        {
            if (readerEndIndex == readerTextLength)
            {
                readerEstPages = readerNumPages = readerPageNum;
                //  dbg("... exact value = "+readerPageNum);
            }
            else                                //  estimate number of pages
            {
                readerEstPages = Math.round(0.5f
                    + (float)readerPageNum
                        * (float)readerTextLength
                        / (float)readerEndIndex);
            //  dbg("... est'd value = "+readerEstPages);
            }
        }
        else
        {
            readerEstPages = readerNumPages;    //  exact value is known
        //  dbg("... exact value = "+readerNumPages);
        }
    }

    public final int getFontIndex(Font itsFont, int itsDefault)
    {
        int itsIndex = 0, itsSize = itsFont.getSize();

        while (itsIndex < readerFontSize.length
            && itsSize != readerFontSize[itsIndex]) itsIndex++;

        if (itsIndex == readerFontSize.length)
            itsIndex = readerSizeIndex;                    //  default

        return itsIndex;
    }

    public final boolean imageUpdate(Image img, int flags, int x, int y, int w, int h)
    {
    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").imageUpdate(flags
= "+flags+")");

        if
        (   (   flags
            &   (   ImageObserver.ALLBITS
                |   ImageObserver.FRAMEBITS
                )
            ) != 0)     //  this image is complete (at last)
        {
            Image pageImage = (Image)readerPageImage.elementAt(readerPageNum - 1);
            if (pageImage != null)
                pageImage.flush();
            readerScreenDirty = true;
            readerPageImage.setElementAt(null, readerPageNum - 1);
            setReaderRect();
            clearRowElements();
            repaintReaderScreen();
            this.repaint();

            return false;
        }
        else
            return true;
    }

    public int indent(boolean right)
    {
    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").indent("+right+")"
);

        if (right)
        {
            if (rightIndent)
            {
                return 2*readerEnWidth;
            }
        }
        else
        {
            if (unorderList)
            {
                int tabs = listAccount.size() > 1?listAccount.size() - 1:0;
                return 2*(tabs + (termDefinition?1:0))*readerEnWidth;
            }
            if (orderedList)
            {
                return 2*listAccount.size()*readerEnWidth;
            }
            if (leftIndent)
            {
                return 2*readerEnWidth;
            }
        }

        return 0;
    }

    public Insets insets()
    {
    //  dbg("_p_tb_db("+theTableNumber+").insets()");

        return insets;
    }

    public final void loadPageData(int pageIndex)
    {
    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").loadPageData("+pag
eIndex+"): readerPageStart.size() = "+readerPageStart.size()+", readerEndIndex =
"+readerEndIndex+", lineImages.size() = "+lineImages.size()+"; listAccount.size() =
"+listAccount.size());

        Image pageImage  =   (Image)    readerPageImage.elementAt(pageIndex);
        listAccount      =   (Vector)   readerListAccts.elementAt(pageIndex);
        linkHeads        =   (Vector)   readerLinkHeads.elementAt(pageIndex);
        linkRects        =   (Vector)   readerLinkRects.elementAt(pageIndex);
        zoomHeads        =   (Vector)   readerZoomHeads.elementAt(pageIndex);
        zoomRects        =   (Vector)   readerZoomRects.elementAt(pageIndex);
        pageComps        =   (Vector)   readerPageComps.elementAt(pageIndex);
        readerStartIndex = ((Integer)
    readerPageStart.elementAt(pageIndex)).intValue();

        hyperHead        =   (URL)      readerLinkStart.elementAt(pageIndex);
        this.restoreStyle   ((Integer)  readerTypeStart.elementAt(pageIndex));
        this.setFont        ((Font)     readerFontStart.elementAt(pageIndex)
                            ,(Color)    readerTintStart.elementAt(pageIndex));

        if (pageImage != null)
            readerScreen = pageImage;
    }

    public final Dimension minimumSize()
    {
    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").minimumSize("+mini
mumSize.width+","+minimumSize.height+")");

        return minimumSize;
    }

    public boolean mouseDown(Event evt, int x, int y)
    {
    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").mouseDown("+x+","+
y+")");

        if
        (   !readerFrame
        &&  zoomable
        &&  readerContent.length() > 0
        )
        {
            Component ancestor = getParent();
            if (ancestor != null)
            {
                if (ancestor instanceof ez_html)
                {
                    return false;
                }
                else
                if (ancestor instanceof _p_tc_db)
                {
                    while (ancestor != null)
                    {
                        if (ancestor instanceof _f_tb_db &&
((_f_tb_db)ancestor).justDragging)
                        {
                            return false;
                        }
                        ancestor = ancestor.getParent();
                    }
                }
            }
            ez_HTML.showStatus(ZN);
        }
        else
        for (int i = 0; i < zoomRects.size(); i++)
        {
            if (zoomRects.elementAt(i) instanceof Rectangle
            &&  ((Rectangle)zoomRects.elementAt(i)).inside(x,y))
            {
                if (zoomHeads.elementAt(i) instanceof String)
                {
                    ez_HTML.showStatus(ZI);
                }
                return true;
            }
        }

        return super.mouseDown(evt, x, y);
    }

    public boolean mouseMove(Event evt, int x, int y)
    {
        for (int i = 0; i < linkRects.size(); i++)
            if (linkRects.elementAt(i) instanceof Rectangle
            &&  linkHeads.elementAt(i) instanceof URL
            &&  ((Rectangle)linkRects.elementAt(i)).inside(x,y))
            {
                ez_HTML.showStatus(((URL)linkHeads.elementAt(i)).toString());
                return true;
            }

        for (int i = 0; i < zoomRects.size(); i++)
            if (zoomRects.elementAt(i) instanceof Rectangle
            &&  zoomHeads.elementAt(i) instanceof String
            &&  ((Rectangle)zoomRects.elementAt(i)).inside(x,y))
            {
                ez_HTML.showStatus(ZM);
                return true;
            }

        if
        (   zoomable
        &&  readerContent.length() > 0
        )
        {
            ez_HTML.showStatus(ez_HTML.ZM);
            return true;
        }
        ez_HTML.showStatus(_p_rs_db.MT);

        return super.mouseMove(evt, x, y);
    }

    public boolean mouseUp(Event evt, int x, int y)
    {
```

```
    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").mouseUp() ");

        for (int i = 0; i < linkRects.size(); i++)
        {
            if (linkRects.elementAt(i) instanceof Rectangle
            &&  linkHeads.elementAt(i) instanceof URL
            &&  ((Rectangle)linkRects.elementAt(i)).inside(x,y))
            {
                ez_HTML.getAppletContext().showDocument
                (   (URL)linkHeads.elementAt(i)
                ,   "_blank"                        //  opens new page in (new, if
cooperative) browser window
                );
                return true;
            }
        }

        if
        (   !readerFrame
        &&  zoomable
        &&  readerContent.length() > 0
        )
        {
            String title = null;
            Component ancestor = getParent();
            if (ancestor != null)
            {
                if (ancestor instanceof ez_html)
                {
                    return super.mouseDown(evt, x, y);
                }
                else
                if (ancestor instanceof _p_tc_db)
                {
                    boolean findTitle = true;
                    while (ancestor != null)
                    {
                        if (ancestor instanceof _p_tb_db && findTitle)
                        {
                            title = ((_p_tb_db)ancestor).theCaption;
                            findTitle = false;
                        }
                        if (ancestor instanceof _f_tb_db &&
((_f_tb_db)ancestor).justDragging)
                        {
                            return super.mouseDown(evt, x, y);
                        }
                        ancestor = ancestor.getParent();
                    }
                }
            }
        //  dbg("... making new _f_rs_db");

            ez_HTML.showStatus(ZW);
            _f_rs_db panelFrame = new _f_rs_db
            (   title
            ,   readerContent
            ,   ez_HTML
            ,   15                              //  max columns
            ,   true                            //  present markers
            ,   true                            //  present footer
            );
            return true;
        }
        else
        for (int i = 0; i < zoomRects.size(); i++)
        {
            if (zoomRects.elementAt(i) instanceof Rectangle
            && ((Rectangle)zoomRects.elementAt(i)).inside(x,y))
            {
                if (zoomHeads.elementAt(i) instanceof String)
                {
                    ez_HTML.showStatus(ZX);
                    _o_im_db.zoomImage((String)zoomHeads.elementAt(i), ez_HTML);
                }
                return true;
            }
        }

        return super.mouseUp(evt, x, y);
    }

    public final void nextColumn()
    {
    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").nextColumn(): old
colNum = "+colNum);

        colNum++;                                       //  finished another column
        lastColumn = colNum == numColumns - 1;

        yb = readerRect.y;                              //  local baseline of first
line
        xb += colSpacing;                               //  local left edge of the
column area
        te += colSpacing;                               //  local end of text
        xe += colSpacing;                               //  local right edge of the
column area
    }

    public final void paint(Graphics g)
    {
    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").paint(g):
isEnabled = "+isEnabled()+", isShowing = "+isShowing()+", isValid = "+isValid()+",
isVisible = "+isVisible());

        if (readerScreenDirty)
        {
            if (readerScreen != null)
                readerScreen.flush();

            readerScreen = createImage(this.size().width, this.size().height);

            Graphics q = this.readerScreen.getGraphics();
            paintScreen(q);     //  adds components (paintArea) for this page & saves
them
            q.dispose();

            readerScreenDirty = false;
            readerPageImage.setElementAt(readerScreen, readerPageNum - 1);
            this.invalidate();
            this.validate();
        }
        else if (this.countComponents() != pageComps.size())    //  restore components
        {
        //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").paint(g):
removeAll() ");
            this.removeAll();
            for (int i = 0; i < pageComps.size(); i++)
            {
                _p_tb_db table = (_p_tb_db)pageComps.elementAt(i);
            //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").paint(g): adding
table "+table.theTableNumber);

                this.add(table);
            //  table.show();   //  just in case it was hidden
            }
        }

        if (g != null)
        {
            g.drawImage(readerScreen,0,0,this);
        //  this.validate();
        }
    }

    private final int paintArea                               //  returns string
index of first non-drawn character
    (   Graphics g
    ,   int beginIndex
    ,   String textContent
    ,   int numCols
    )
    {
    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").paintArea(beginInd
ex = "+beginIndex+")");
    //  long time = System.currentTimeMillis();

        int endText = textContent.length();
        endIndex = beginIndex;

        numColumns = numCols;
        colNum = 0;
        int colMargin = 2*readerEnWidth;                      //  2 n-spaces between
cols
        colWidth = (readerRect.width - (numColumns-1)*colMargin)/numColumns;
        colSpacing = colWidth + colMargin;

        String nextWord = MT;

        xb = readerRect.x;                                    //  local left edge of
the column area
        te = xb + indent(false);                              //  end position of
this line's words & graphics
        xe = xb + colWidth;                                   //  local right edge
of the column area

        nextWidth = 0;

        yb = readerRect.y;                                    //  local starting top
of first line
        ye = readerRect.y + readerRect.height;                //  local maximum
bottom of last line
    //  dbg("... yb = "+yb+"(initially), ye = "+ye);

        lastColumn = colNum == numColumns - 1;
        lastRow = false;
        controlSize = textHeight*2/3;

        if (markings)
            setMarkers(g, controlSize);

        clearRowElements();

        StringTokenizer tagBlocks = new StringTokenizer        //  break up the text
into normal tagBlocks
            (textContent.substring(beginIndex),PF,true);       //  final true returns
token on next call

```

```
        draw: while (tagBlocks.hasMoreTokens() && !(yb > ye && lastColumn))
        {                                                       //  tag block to
follow
            String tagBlock = tagBlocks.nextToken(PF);          //  [body text],[<tag
text>body text],...,[<tag text>body text]

        //  handle all html tag instructions

            if (!plaintext && tagBlock.equals(PF))              //  an html "<" (PF)
tag prefix?
            {
                String tagText = tagBlocks.nextToken(SF);       //  should be tag text

                if (tagBlocks.hasMoreTokens())
                    tagBlock = tagBlocks.nextToken();           //  should be ">" (SF)
tag suffix
                else
                    tagBlock = MT;                              //  syntax error

                if (tagBlock.equals(SF))
                {
                    int endIncr = addTag(tagText, g);           //  accounts for any
paintRow() indexing
                    if (endIncr < 0)
                    {
                        endIndex -= endIncr;                    //  account for what
was actually done
                        break draw;                             //  incomplete tag
management (no room for it)
                    }
                    else
                    {
                        endIndex += endIncr;                    //  been there, done
that, what's next
                    }
                    tagBlock = MT;                              //  tagText handled,
transfer text capture
                }
                else                                            //  recover from
syntax error
                {
                    tagBlock = PF + tagText + tagBlock;
                }
            }

        //  handle all html text content

            if (!tagBlock.equals(PF) && !tagBlock.equals(MT))   //  not an html tag?
            {
                if (!addBlock(tagBlock,g,true)) break draw;
            }
        }   //  end of draw                                     //  no more tagBlocks
        endIndex += paintRow(g, 0);                             //  try to punch out
pending line elements

        while (endIndex < endText
        && textContent.substring(endIndex++).startsWith(BL));   //  skip blanks

        savePageData(readerPageNum, endIndex);                  //  for the next page

        estimateNumPages();
        if (markings)
            paintMarkers(g, controlSize);

    //  dbg("area paint duration = "+(System.currentTimeMillis()-time)+" ms.");

        return endIndex;
    }

    public final void paintMarkers(Graphics g, int controlSize)
    {
        if (g == null) return;

        g.setColor(getForeground());

        if (!readerFrame)                                          //  right arrow
        {
            int x = aheadRect.x + aheadRect.width - 2
            ,   y = aheadRect.y + aheadRect.height/2;
            for (int arrow = 0; arrow < controlSize; arrow++)
            {
                g.drawLine(x - arrow,y + arrow/2,x - arrow,y - arrow/2);
            }
        }
    }

    public final synchronized int paintRow(Graphics g, int retain)
    {
    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").paintRow(g,
"+retainCount+") @ ("+xb+","+yb+"), left indent = "+indent(false)+", right indent =
"+indent(true)+"; stack("+dataValues.size()+")");

        int retainCount = Math.max(0,retain);         //  ensure reasonable results
        int maxAscent = 0;                            //  measure the max baseline
ascent for output
        int maxDescent = 0;                           //  measure the max baseline
descent for output
        int lineWidth = 0;                            //  measure the full width of this
output
        int indexIncr = 0;                            //  measure the number of
characters represented by this line

        for (int i = 0; i < dataValues.size() - retainCount; i++)
        {
            Rectangle d = (Rectangle)dataValues.elementAt(i);

            maxAscent = Math.max(maxAscent, d.y);
            maxDescent = Math.max(maxDescent, d.height);
            lineWidth += d.width;
        }
        yb += maxAscent + maxDescent;

        while (yb > ye)                               //  start a new column?
        {
        //  dbg("... yb > ye");
            if (lastColumn) return indexIncr;         //  no more columns left to fill
            nextColumn();
            yb += maxAscent + maxDescent;
        }

        int itsStyle = LEFT;
        if (styleCodes.size() > 0)
        {
            if
            (   g != null
            &&  wordColors.size() > 0
            &&  charsFonts.size() > 0
            )
            {
                g.setColor((Color)wordColors.firstElement());
                g.setFont((Font)charsFonts.firstElement());
            }

            restoreStyle(itsStyle = ((Integer)styleCodes.firstElement()).intValue());
        }

        int x = xb + indent(false) + textAlign*(effectiveWidth() - lineWidth)/2;
        int y = yb - maxDescent;                    //  go up to baseline and draw
above & below it

        while
        (   lineImages.size() > retainCount
        &&  dataValues.size() > retainCount
        &&  wordBlocks.size() > retainCount
        &&  hyperLinks.size() > retainCount
        &&  styleCodes.size() > retainCount
        &&  wordColors.size() > retainCount
        &&  charsFonts.size() > retainCount
        )      //  draw all unretained elements
        {
            Rectangle d = (Rectangle)dataValues.firstElement();
            Image anOffImage = (Image)lineImages.firstElement();
            String wordBlock = null;
            if (wordBlocks.firstElement() instanceof String)
                wordBlock = (String)wordBlocks.firstElement();

        //  dbg("... anOffImage "+(anOffImage!=null?"!":"=")+"= null; wordBlock
="+(wordBlock!=null?(wordBlock==MT?"= MT":" "+wordBlock):"= null")+";");

            if (anOffImage != null              //  either a graphic or a table image

            &&  wordBlock != null)
            {
            //  dbg("... drawImage()");
                Image origImage = ez_HTML.getImageDB().fetchImage(wordBlock);
                if (origImage.getWidth(ez_HTML) > anOffImage.getWidth(ez_HTML))
                {
                    zoomHeads.addElement(wordBlock);
                    zoomRects.addElement(new Rectangle(x, y - d.y,
anOffImage.getWidth(ez_HTML), anOffImage.getHeight(ez_HTML)));
                }

                if (g != null) g.drawImage(anOffImage, x, y - d.y, this);
                anOffImage.flush();
            }
            else                                //  table reference stored in
wordBlocks
            if (wordBlocks.firstElement() instanceof _p_tb_db)
            {
                _p_tb_db theTable = (_p_tb_db)wordBlocks.firstElement();
                theTable.validate();
                maxWidth = Math.max(maxWidth, theTable.preferredSize().width);
                maxHeight = Math.max(maxHeight, theTable.preferredSize().height);
            }
            else if (MT == wordBlock)           //  controlling tag
            {
                if (g != null)
                {
                    g.setColor((Color)wordColors.firstElement());
                    g.setFont((Font)charsFonts.firstElement());
                }

                restoreStyle(itsStyle =
((Integer)styleCodes.firstElement()).intValue());
            //  dbg("... update: style("+itsStyle+"): newListItem = "+newListItem+";
unorderList = "+unorderList+", orderedList = "+orderedList);
            }
            else if (wordBlock != null)
            {
            //  dbg("... drawString(["+wordBlock+"],x,y); itsStyle = "+itsStyle);

                if (newListItem && listAccount.size() > 0)
                {
                    ((Point)listAccount.firstElement()).x++;    //  bookkeeping count
                    if (unorderList)
                    {
                        _o_li_db.drawBullet(this, x, y, g);
                    }
                    if (orderedList)
                    {
                        _o_li_db.drawLabel(this, x, y, g);
                    }
                }

                boolean doUnderline = (itsStyle & UNDERLINE) != 0 || (itsStyle &
HYPERLINK) != 0;
                boolean doLineStrike = (itsStyle & STRIKETHRU) != 0;

                int shift = 0;

                if ((itsStyle & SUBSCRIPT) != 0)
                    shift += (textHeight+3)/4 - 1;
                if ((itsStyle & SUPSCRIPT) != 0)
                    shift -= (textHeight+3)/4 - 1;

                if (g != null)
                {
                    if (doUnderline)
                        g.drawLine(x, y + 2 + shift, x + d.width, y + 2 + shift);
                    if (doLineStrike)
                        g.drawLine(x, y - (textHeight+3)/4 + shift, x + d.width, y -
(textHeight+3)/4 + shift);

                    g.drawString(wordBlock, x, y + shift);
                }

                if
                (   hyperLinks.size() > 0
                &&  hyperLinks.firstElement() != null
                )
                {
                //  if (debug && g != null) g.drawRect(x, y - d.y + shift, d.width +
readerEnWidth/2, d.y + d.height);
                    linkHeads.addElement(hyperLinks.firstElement());
                    linkRects.addElement(new Rectangle(x, y - d.y + shift, d.width +
readerEnWidth/2, d.y + d.height));
                }
            }

            x += d.width;
            indexIncr += d.x;

            if
            (   lineImages.size() > 0
            &&  dataValues.size() > 0
            &&  wordBlocks.size() > 0
            &&  hyperLinks.size() > 0
            &&  styleCodes.size() > 0
            &&  wordColors.size() > 0
            &&  charsFonts.size() > 0
            )
            {
                lineImages.removeElementAt(0);
                dataValues.removeElementAt(0);
                wordBlocks.removeElementAt(0);
                hyperLinks.removeElementAt(0);
                styleCodes.removeElementAt(0);
                wordColors.removeElementAt(0);
                charsFonts.removeElementAt(0);
            }
            else
            {
        /*      dbg
                (   "unexpected size(s) in paintRow("+retain+"):\n"
                +   "lineImages.size() " + lineImages.size() + "\n"
                +   "dataValues.size() " + dataValues.size() + "\n"
                +   "wordBlocks.size() " + wordBlocks.size() + "\n"
```

```
                    +   "hyperLinks.size() " + hyperLinks.size() + "\n"
                    +   "styleCodes.size() " + styleCodes.size() + "\n"
                    +   "wordColors.size() " + wordColors.size() + "\n"
                    +   "charsFonts.size() " + charsFonts.size() + "\n"
                    );
        */  }
        }
        maxWidth = Math.max(maxWidth, lineWidth);
        maxHeight = Math.max(maxHeight, yb - readerRect.y);

        savePageData(readerPageNum, endIndex + indexIncr);      //  for the next page

    //  dbg("... final stack("+dataValues.size()+")");
        return indexIncr;
    }

    private final synchronized void paintScreen(int width)      //  test (null) paint
for sizing
    {
    //  dbg("paintScreen("+width+")");

        Dimension s = getToolkit().getScreenSize();
        int wide = width + insets.left + insets.right + 2;      //  one column wide
        int tall = s.height;                                    //  super column
height
        Rectangle r = new Rectangle
        (   0
        ,   0
        ,   Math.min(wide, s.width)                             //  set to indicated
width
        ,   tall
        );
        paintScreen(null, r);
    }

    private final synchronized void paintScreen(Graphics g)
    {
    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").paintScreen(g):
getBackground() = "+getBackground());

        Rectangle r = bounds();                                 //  set to full area
(this.bounds())
        paintScreen(g, r);
    }

    private final synchronized void paintScreen(Graphics g, Rectangle r)
    {
    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").paintScreen(g):
getBackground() = "+getBackground());

        setReaderRect(r);                               //  set to full area
(this.bounds())

        if (g != null)
        {
            g.setColor(getBackground());
            g.fillRect(0, 0, r.width, r.height);
            for (int i = 0; i < border; i++)
                g.draw3DRect
                (   i + spacing/2
                ,   i + spacing/2
                ,   r.width  - 1 - spacing - 2*i
                ,   r.height - 1 - spacing - 2*i
                ,   false
                );
        //  background fill, if requested

            Image background = ez_HTML.getBackImage();
            if (background != null)
            {
                Point p = this.location();
                int w = background.getWidth(this);
                int h = background.getHeight(this);
                if (w > 0 && h > 0)
                {
                    int x = this.size().width;
                    int y = this.size().height;
                    for (int i = 0; i < x; i += w)
                    {
                        for (int j = 0; j < y; j += h)
                        {
                            g.drawImage(background,i-p.x,j-p.y,this);
                        }
                    }
                }
            }

        }

        clearPage();
        maxHeight = 0;

        int numCols = calculateReaderColumns();
        if (numCols != readerNumCols)
        {
            readerNumCols = numCols;
            resetReaderView();
        }

        if (readerFrame)
            readerLeading = 2;                          //  make it more comfortable
        else
            readerLeading = 0;                          //  make it more compact

    //  paint the content area

        loadPageData(readerPageNum-1);                  //  for the current page
        markings = readerMarker && !readerFooter;
        readerEndIndex = paintArea
        (   g
        ,   readerStartIndex
        ,   readerContent
        ,   readerNumCols
        );
        readerStartIndex = readerEndIndex - 1;
        estimateNumPages();

        if (preferredSize.width < 0
        ||  preferredSize.height < 0)                   //  set and forget
        {
            Dimension screenSize = getToolkit().getScreenSize();

            preferredSize.width = Math.min
            (   screenSize.width
            ,   Math.max
                (   minimumSize.width
                ,   insets.left + insets.right + 2
                +   maxWidth                            //  single column format
                )
            );

            preferredSize.height = Math.min
            (   screenSize.height
            ,   Math.max
                (   minimumSize.height
                ,   insets.top + insets.bottom + 2
                +   maxHeight * (colNum + 1)
                +   (readerFooter?readerFontSize[readerFootIndex]*3/2:0)
                )
            );
        //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").paintScreen(): new
preferredSize("+preferredSize.width+","+preferredSize.height+"); complete =
"+(readerEndIndex == readerTextLength)+"; maxHeight = "+maxHeight+", readerNumCols =
"+readerNumCols);
        }
    }

    public final Dimension preferredSize()
    {
        return preferredSize(standardLineWidth());
    }

    public final Dimension preferredSize(int prefLineWidth)
    {
        if (preferredSize.width < 0 || preferredSize.height < 0)
        {
            resetReaderView();
            paintScreen(prefLineWidth);
            resetReaderView();
        //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").preferredSize("+pr
eferredSize.width+","+preferredSize.height+") - from null screen layout");

            return preferredSize;
        }
        else
        {
        //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").preferredSize("+pr
eferredSize.width+","+preferredSize.height+") - prior layout");

            return preferredSize;
        }
    }

    public synchronized void removeAll()     //  override, because some browsers fail
here w/o layout manager
    {
        super.removeAll();

        int n = countComponents();
        if (n > 0)
            this.invalidate();
        while (n > 0)
        {
            Component c = getComponent(n-1);
            c.hide();
            c.invalidate();
            remove(c);
            n = countComponents();
        }
    }

    public final void repaintReaderScreen()
    {
    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").repaintReaderScree
n()");

        if (readerPageImage.elementAt(readerPageNum - 1) == null)
        {
            readerScreenDirty = true;
        }
        this.repaint();
    }

    public final synchronized void resetReaderView()
    {
    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").resetReaderView()"
);

        readerStartIndex    = 0;
        readerEndIndex      = 1;
        readerNumPages      = 0;
        readerEstPages      = 1;
        readerPageNum       = 1;

        readerPageImage.removeAllElements();
        readerListAccts.removeAllElements();
        readerLinkHeads.removeAllElements();
        readerLinkRects.removeAllElements();
        readerZoomHeads.removeAllElements();
        readerZoomRects.removeAllElements();
        readerPageComps.removeAllElements();
        readerPageStart.removeAllElements();

        readerLinkStart.removeAllElements();
        readerTypeStart.removeAllElements();
        readerFontStart.removeAllElements();
        readerTintStart.removeAllElements();

        clearPage();
        maxWidth = 0;
        newListItem = false;
        plaintext = false;

        readerScreenDirty = true;

        restoreStyle(0);
        this.setFont(readerScreenFont,getForeground());

        savePageData(0, readerEndIndex);
    }

    public void reshape(int x, int y, int width, int height)
    {
        super.reshape(x, y, width, height);
        resetReaderView();

    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").reshape("+x+",
"+y+", "+width+", "+height+")");

        repaint();
    }

    public final void restoreStyle(Integer theStyleCode)
    {
        restoreStyle(theStyleCode.intValue());
    }

    public final void restoreStyle(int itsStyle)
    {
        textAlign       = (itsStyle & POSITION   );
        underlining     = (itsStyle & UNDERLINE  ) != 0;
        strikethrough   = (itsStyle & STRIKETHRU ) != 0;
        subscript       = (itsStyle & SUBSCRIPT  ) != 0 && (itsStyle & SUPSCRIPT ) == 0;
        supscript       = (itsStyle & SUPSCRIPT  ) != 0 && (itsStyle & SUBSCRIPT ) == 0;
        hyperLink       = (itsStyle & HYPERLINK  ) != 0;
        leftIndent      = (itsStyle & LEFTINDENT ) != 0;
        rightIndent     = (itsStyle & RIGHTINDENT) != 0;
        hidingText      = (itsStyle & HIDDENTEXT ) != 0;
        unorderList     = (itsStyle & UNORDERLIST) != 0;
        orderedList     = (itsStyle & ORDEREDLIST) != 0;
    }

    private final void savePageData(int nextPageIndex, int currEndIndex)
    {
    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(readerContent.length()<30?ZZ.substring(readerContent.length()):MT)+").savePageData("+nextPageIndex+","+currEndIndex+"): new = "+(readerNumPages == 0 &&  nextPageIndex == readerPageStart.size())+", readerEndIndex = "+readerEndIndex+", lineImages.size() = "+lineImages.size()+"; listAccount.size() = "+listAccount.size());

        if (readerNumPages == 0
        &&  nextPageIndex  == readerPageStart.size())    //  the first time reaching this page
        {
            readerPageImage.addElement(null);            //  readerScreen is loaded by paint(g)

            Vector currList = new Vector(listAccount.size());
            for (int i = 0; i < listAccount.size(); i++)
            {
                Point p = (Point)listAccount.elementAt(i);
                currList.addElement(new Point(p.x, p.y));
            }
            listAccount = currList;
            readerListAccts.addElement(listAccount);
            readerLinkHeads.addElement(new Vector());
            readerLinkRects.addElement(new Vector());
            readerZoomHeads.addElement(new Vector());
            readerZoomRects.addElement(new Vector());
            readerPageComps.addElement(new Vector());    //  components are added by placeTable(g)
            readerPageStart.addElement(new Integer(currEndIndex - 1));

            if (nextPageIndex > 0
            &&  lineImages.size() > 0)                   //  unpainted elements in stack
            {
                readerLinkStart.addElement((URL)hyperLinks.firstElement());
                readerFontStart.addElement((Font)charsFonts.firstElement());
                readerTintStart.addElement((Color)wordColors.firstElement());
                readerTypeStart.addElement((Integer)styleCodes.firstElement());
            }
            else
            {
                readerLinkStart.addElement(hyperHead);
                readerFontStart.addElement(textFont);
                readerTintStart.addElement(textColor);
                readerTypeStart.addElement(styleCode());
            }
        }
        else                                             //  update next page initial state vectors
        {
            readerPageStart.setElementAt(new Integer(currEndIndex - 1), nextPageIndex);
            if (nextPageIndex > 0
            &&  lineImages.size() > 0)                   //  unpainted elements in stack
            {
                    readerLinkStart.setElementAt((URL)hyperLinks.firstElement(),
nextPageIndex);
                    readerFontStart.setElementAt((Font)charsFonts.firstElement(),
nextPageIndex);
                    readerTintStart.setElementAt((Color)wordColors.firstElement(),
nextPageIndex);
                    readerTypeStart.setElementAt((Integer)styleCodes.firstElement(),
nextPageIndex);
               }
               else
               {
                    readerLinkStart.setElementAt(hyperHead, nextPageIndex);
                    readerFontStart.setElementAt(textFont, nextPageIndex);
                    readerTintStart.setElementAt(textColor, nextPageIndex);
                    readerTypeStart.setElementAt(styleCode(), nextPageIndex);
               }
          }
     }

     public void setAllFonts()
     {
          readerFooterFont = new Font(FS, Font.PLAIN, readerFontSize[readerFootIndex]);
          readerScreenFont = new Font(FS, Font.PLAIN, readerFontSize[readerSizeIndex]);

          footerTextHeight = readerFontSize[readerFootIndex]*5/4;

          setFont(readerScreenFont, getForeground());
          readerEnWidth = getFontMetrics(readerScreenFont).stringWidth("n");
     }

     public final void setFont(Font f, Color c)
     {
          textColor = c;
          this.setFont(f);
     }

     public final void setFont(Font f)
     {
          textFont = f;
          textMetrics = getFontMetrics(f);
          textHeight = f.getSize()*5/4;
          textHalfSpace = (textHeight + readerLeading)/2;

     //
     dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").setFont():
textHeight = "+textHeight);
     }

     public synchronized final void setFontSizeIndex(int newSizeIndex)
     {
     //
     dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").setFontSize("+newS
izeIndex+"): # comps = "+countComponents());

          readerSizeIndex = newSizeIndex;
          readerFootIndex = readerSizeIndex - 2;

          readerSizeIndex = Math.max(Math.min(readerSizeIndex,readerFontSize.length -
1),0);
          readerFootIndex = Math.max(Math.min(readerFootIndex,readerFontSize.length -
1),0);

          setAllFonts();
     }

     public void setMarkers(Graphics g, int controlSize)
     {
          aheadRect.reshape
```

```
        (   readerRect.x + readerRect.width - 2 - (controlSize+4)*1
        ,   readerRect.y + readerRect.height - 4 - controlSize
        ,   controlSize + 4
        ,   controlSize + 4
        );
    }

    public void setReaderRect()
    {
    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").setReaderRect()");

        setReaderRect(this.bounds());
    }

    public void setReaderRect(Rectangle r)
    {
        readerRect.reshape
        (   insets.left
        ,   insets.top
        ,   r.width  - 1 - insets.left - insets.right
        ,   r.height - 1 - insets.top - insets.bottom
        -   (readerFooter?readerFontSize[readerFootIndex]*3/2:0)
        );
        firstWord = true;

    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").setReaderRect("+r.
x+","+r.y+","+r.width+","+r.height+"):
readerRect.reshape("+readerRect.x+","+readerRect.y+","+readerRect.width+","+readerRect
.height+")");
    }

    public final void setZoomable(boolean zoomable)
    {
        this.zoomable = zoomable;
    }

    public final int standardLineWidth()
    {
        int numChar = 30;           //  the control parameter: min # of n's per line
of text
        int stndLineWidth = numChar * readerEnWidth;

    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").standardLineWIdth(
) = "+stndLineWidth);

        return stndLineWidth;
    }

    public final boolean startNewParagraph(int halfSpaces)
    {
        if (yb > readerRect.y)                  //  unless we're at the top
        {
            yb += halfSpaces * textHalfSpace;   //  drop down to next baseline
            if (yb > ye)
            {                                   //  not enough room in this (partial?)
column
                if (lastColumn) return false;   //  no more columns left to fill
                nextColumn();
            }
        }
        te = xb + indent(false);                //  start a new line
        maxHeight = Math.max(maxHeight, yb - readerRect.y);

        return true;
    }

    public void stepFontSize(boolean smaller)
    {
    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").stepFontSize(small
er = "+smaller+")");

        if (smaller)    readerSizeIndex--;
        else            readerSizeIndex++;

        setFontSizeIndex(readerSizeIndex);
    }

    public final Integer styleCode()
    {
        return  new Integer
            (   textAlign
            +   ((underlining)              ?   UNDERLINE   :0)
            +   ((strikethrough)            ?   STRIKETHRU  :0)
            +   ((subscript && !supscript)  ?   SUBSCRIPT   :0)
            +   ((supscript && !subscript)  ?   SUPSCRIPT   :0)
            +   ((hyperLink)                ?   HYPERLINK   :0)
            +   ((leftIndent)               ?   LEFTINDENT  :0)
            +   ((rightIndent)              ?   RIGHTINDENT :0)
            +   ((hidingText)               ?   HIDDENTEXT  :0)
            +   ((unorderList)              ?   UNORDERLIST :0)
            +   ((orderedList)              ?   ORDEREDLIST :0));
    }

    public void update(Graphics g)
    {
    //  fillRect is NOT performed here to eliminate blank screen boredom during
offscreen drawing

    //  g.setColor(getBackground());
    //  g.fillRect(0, 0, width, height);
    //  g.setColor(getForeground());

        paint(g);
    }

    public _p_rs_db
    (   String readerContent
    ,   ez_html ez_HTML
    ,   _f_rs_db frame
    ,   boolean readerFrame
    ,   int maxColumns
    ,   boolean marker
    ,   boolean footer
    ,   boolean zoomable
    )
    {
        this
        (   readerContent
        ,   ez_HTML
        ,   frame
        ,   readerFrame
        ,   maxColumns
        ,   marker
        ,   footer
        ,   readerFrame?new Color(239,239,239):ez_HTML.getBackColor()
        ,   readerFrame?Color.black:ez_HTML.getForeColor()
        ,   0               //  no border
        ,   10              //  no padding
        ,   0               //  no spacing
        ,   zoomable        //  zoomability
        );
    }

    public _p_rs_db
    (   String readerContent
    ,   ez_html ez_HTML
    ,   _f_rs_db frame
    ,   boolean readerFrame
    ,   int maxColumns
    ,   boolean marker
    ,   boolean footer
    ,   Color panelBg
    ,   Color panelFg
    ,   int border
    ,   int padding
    ,   int spacing
    ,   boolean zoomable
    )
    {
    //
    dbg("_p_rs_db("+readerContent.substring(0,Math.min(30,readerContent.length()))+(re
aderContent.length()<30?ZZ.substring(readerContent.length()):MT)+").<init>\n"+readerCo
ntent);

        this.setLayout(null);

        this.ez_HTML = ez_HTML;
        this.frame = frame;
        this.readerFrame = readerFrame;
        this.readerHeader = ez_HTML.getTitle() != null;
        this.readerContent = readerContent;
        this.readerTextLength = readerContent.length();
        this.charEntConverter = new _o_nt_db();
        this.maxColumns = Math.max(1, maxColumns);
        this.readerMarker = marker;
        this.readerFooter = footer;
        setBackground(panelBg);
        setForeground(panelFg);
        this.border = border;
        this.padding = padding;
        this.spacing = spacing;
        this.zoomable = zoomable;

        insets.top      =
        insets.left     = border + padding + spacing/2;
        insets.bottom   =
        insets.right    = border + padding + spacing - spacing/2;

        setAllFonts();
        if (readerFrame)
        {
            stepFontSize(false);                                    //  larger font for
reader screens
        }
        readerEnWidth = getFontMetrics(readerScreenFont).stringWidth("n");
    }
}

/**
 *  %W% %E% Everett Stoub
 *
 *  Copyright (c) '96-'97 ION Web Products, Inc. All Rights Reserved.
 *
 *  a frame for presentinginteractive multicolumn text and graphics
 *
 *  @author Everett Stoub %I%, %G%
 */
import java.awt.*;
import java.applet.*;
import java.net.*;
import java.io.*;
import java.lang.*;
import java.util.*;

public final class _f_rs_db extends Frame implements Runnable
{
    public ez_html      ez_HTML = null;

    public _p_rs_db     panel = null;
    public _p_pm_db     locator = null;
    public Panel        control = null;
    public Panel        subpanel = null;

    public Thread       runner = null;
    public int          messageNumber = 0;
    public Label        messageLabel = null;
    public boolean      stopped = false;
    public String[]     message =
                        {   "About: EZ HTML Version 0.95a1..."
                    //  ,   "...Certified 100% Pure Java"
                        ,   "...More info: 888.ION.JAVA"
                        ,   "...URL: www.ionsystems.com"
                        ,   "...Created by Everett Stoub"
                        ,   "...(c)'97, ION Systems, Inc."
                        };

    private String buttonName[] =
    {   "Next"
    ,   "Previous"
    ,   "Font +"
    ,   "Font -"
//  ,   "Home"
    };
    public String CP = "Center";

    public void start()
    {
        if (runner == null)
        {
            runner = new Thread(this);
            runner.start();
        }
    }

    public void stop()
    {
        if (runner != null && runner.isAlive())
            runner.stop();
        runner = null;
    }

    public void run()
    {
        while (runner != null)
        {
            if (messageNumber < message.length)
                messageLabel.setText(message[messageNumber++]);
            try
            {
                Thread.sleep(3000);
            }
            catch (InterruptedException e) {}
        }
    }

    public boolean action(Event e, Object o)
    {
        if (e.target instanceof Button)
        {
            if (o == (buttonName[0]))   doViewerRight();        else
            if (o == (buttonName[1]))   doViewerLeft();         else
            if (o == (buttonName[2]))   doViewerSize(false);    else
            if (o == (buttonName[3]))    doViewerSize(true); /*   else
            if (o == (buttonName[4]))    doViewerHome();      */

            messageLabel.setText(message[messageNumber = 0]);
        }

        return false;
    }

    private final void dbg(String   s)
    {
        if (debug) System.out.println(s);
    }
    private boolean debug = false;       //  Print debugging info?

/*  public final void doViewerHome()
    {
    //  dbg("_f_rs_db.doViewerHome()");

        if (panel.readerPageNum > 1)
        {
            subpanel.remove(panel);
            panel.removeAll();
            panel.readerPageNum = 1;
            panel.loadPageData(panel.readerPageNum-1);
            subpanel.add(CP, panel);
        }
        panel.repaintReaderScreen();
        if (locator != null)
        {
            locator.repaint();
        }
    }

*/  public final void doViewerLeft()
    {
    //  dbg("_f_rs_db.doViewerLeft()");

        if (panel.readerPageNum > 1)
        {
            subpanel.remove(panel);
            panel.removeAll();
            panel.readerPageNum--;
            panel.loadPageData(panel.readerPageNum-1);
            subpanel.add(CP, panel);
        }
        panel.repaintReaderScreen();
        if (locator != null)
        {
            locator.repaint();
        }
    }

    public final void doViewerRight()
    {
    //  dbg("_f_rs_db.doViewerRight(): readerPageNum = "+panel.readerPageNum+",
readerEstPages = "+panel.readerEstPages);

        if (panel.readerPageNum < panel.readerEstPages)
        {
            panel.readerPageNum++;
        }
        else if (panel.readerNumPages == 0)
        {
            panel.readerNumPages = panel.readerPageNum;
        }
        subpanel.remove(panel);
        panel.removeAll();
        panel.loadPageData(panel.readerPageNum-1);
        panel.repaintReaderScreen();
        subpanel.add(CP, panel);
        if (locator != null)
        {
            locator.repaint();
        }
    }

    public final void doViewerSize(boolean smaller)
    {
    //  dbg("_f_rs_db.doViewerSize(boolean "+smaller+")");

        panel.doViewerSize(smaller);
        subpanel.remove(panel);
        panel.removeAll();
        if (locator != null)
        {
            subpanel.remove(locator);
            panel.resize
            (   panel.size().width
            ,   panel.size().height
            +   locator.size().height
            -   locator.preferredSize().height
            );
            locator.move
            (   locator.location().x
            ,   locator.location().y
            +   locator.size().height
            -   locator.preferredSize().height
            );
            locator.resize
            (   locator.size().width
            ,   locator.preferredSize().height
            );
            subpanel.add("South", locator);
            locator.repaint();
        }
        subpanel.add(CP, panel);
    }

    public synchronized void dispose()
    {
        this.stop();

        super.dispose();
    }

    public final boolean handleEvent(Event evt)
    {
    //  dbg("_f_rs_db.handleEvent("+evt+")");

        switch (evt.id)
        {
            case Event.WINDOW_EXPOSE:       panel.requestFocus();        break;
            case Event.WINDOW_DEICONIFY:    this.show();                 break;
            case Event.WINDOW_ICONIFY:      this.hide();                 break;
            case Event.WINDOW_DESTROY:      ez_HTML.disposeFrame(this); break;
        }
        return super.handleEvent(evt);
    }

    public final boolean keyDown(Event evt, int key)     //  note:  keyUp is not
utilized at all (missing in JDK 1.02 on Mac)
    {
    //  dbg("_f_rs_db.keyDown("+evt+", "+key+")");

        boolean down = key == Event.DOWN;

        switch (key)
        {
        //  case Event.HOME:    doViewerHome();     break;
```

```
            case Event.PGDN:
            case Event.RIGHT:   doViewerRight();    break;
            case Event.PGUP:
            case Event.LEFT:    doViewerLeft();     break;
            case Event.UP:
            case Event.DOWN:    doViewerSize(down); break;

            default: return false;
        }
        messageLabel.setText(message[messageNumber = 0]);

        return true;
    }

    public final Dimension minimumSize()
    {
        Dimension s = getToolkit().getScreenSize();
        if (debug)
        {
            s.width  /= 2;
            s.height /= 2;
        }
        else
        {
            s.height -= 50; // hack for menus
        }

        Dimension b = control.minimumSize();
        Dimension r = subpanel.minimumSize();
        Dimension f = new Dimension(Math.max(b.width, r.width), b.height + r.height);

        double fArea = f.height * f.width;
        double sArea = s.height * s.width;
        if (fArea > sArea)                                  //  more than the screen area
        {
            f.width = s.width;
            f.height = s.height;
        }
        else                                                //  adjust dimensions to
screen aspect
        {
            f.width  = (int)Math.round(Math.sqrt(fArea * s.width  / s.height));
            f.height = (int)Math.round(Math.sqrt(fArea * s.height / s.width ));
        }
        s.width  = Math.min(s.width,  f.width );
        s.height = Math.min(s.height, f.height);

        return s;
    }

    public boolean mouseDown(Event evt, int x, int y)
    {
        if (runner != null && runner.isAlive())
        {
            if (runner != null && runner.isAlive())
            {
                if (stopped)
                {
                    messageNumber = 0;
                    runner.resume();
                }
                else
                {
                    runner.suspend();
                }
                stopped = !stopped;
            }
            else
            {
                stopped = false;
                runner = new Thread(this);
                runner.start();
            }
        }

        return false;
    }

    public final Dimension preferredSize()
    {
        Dimension s = getToolkit().getScreenSize();
        if (debug)
        {
            s.width  /= 2;
            s.height /= 2;
        }
        else
        {
            s.height -= 50; // hack for menus
        }

        Dimension b = control.preferredSize();
        Dimension r = subpanel.preferredSize();
        Dimension f = new Dimension(Math.max(b.width, r.width), b.height + r.height);

        double fArea = f.height * f.width;
        double sArea = s.height * s.width;
        if (fArea > sArea)                              //  more than the screen area
        {
            f.width = s.width;
            f.height = s.height;
        }
        else                                            //  adjust dimensions to
screen aspect
        {
            f.width  = (int)Math.round(Math.sqrt(fArea * s.width  / s.height));
            f.height = (int)Math.round(Math.sqrt(fArea * s.height / s.width ));
        }
        s.width  = Math.min(s.width,  f.width );
        s.height = Math.min(s.height, f.height);

        return s;
    }

    public synchronized void reshape(int x, int y, int width, int height)
    {
        messageLabel.setText(message[messageNumber = 0]);
        super.reshape(x, y, width, height);
    }
    public void update(Graphics g)
    {
    //  g.setColor(getBackground());
    //  g.fillRect(0, 0, width, height);
    //  g.setColor(getForeground());

        paint(g);
    }

    public _f_rs_db
    (   String title
    ,   String readerContent
    ,   ez_html ez_HTML
    ,   int maxColumns
    ,   boolean marker
    ,   boolean footer
    )
    {
        super(title);

    //  dbg("_f_rs_db.<init>");
```

```

        this.ez_HTML = ez_HTML;
        this.setLayout(new BorderLayout());

        subpanel = new Panel();
        subpanel.setLayout(new BorderLayout());
        this.add(CP, subpanel);

        this.panel = new _p_rs_db
        (   readerContent
        ,   ez_HTML
        ,   this
        ,   true
        ,   maxColumns
        ,   false   //  marker
        ,   false   //  footer
        ,   false   //  not zoomable
        );
        this.panel.insets.left  += 10;
        this.panel.insets.right += 10;

        subpanel.add(CP, panel);

        if (footer)
        {
            locator = new _p_pm_db(panel, this);
            subpanel.add("South", locator);
        }

    //  control.add(new Label("ION EZ HTML v0.94a6 ")); //  [Δn-major].[Δn-minor][Δn-
tweak][Δa-fix]
        control = new Panel();
        control.setBackground(Color.gray);
        control.setLayout(new FlowLayout(FlowLayout.LEFT));
        this.add("South", control);

        for (int i = 0; i < buttonName.length; i++)
        {
            Button b = new Button(buttonName[i]);
            b.setBackground(Color.white);
            control.add(b);
        }
        boolean mac = System.getProperty("os.name").equals("macos");
    //  control.add(new Label((mac?"©":"(c)")+"'97, ION Systems, Inc. 888.ION.JAVA"));
        messageLabel = new Label(message[0]);
        control.add(messageLabel);

        this.pack();

    //  dbg("_f_rs_db.init<>: this.size() =
("+this.size().width+","+this.size().height+"), panel.size() =
("+panel.size().width+","+panel.size().height+")");

        this.show();
        this.start();
    }
}

/**
 *  %W% %E% Everett Stoub
 *
 *  Copyright (c) '96-'97 ION Web Products, Inc. All Rights Reserved.
 *
 *  a simple panel to display page position and interactive markers
 *
 *  @author Everett Stoub %I%, %G%
 */
import java.awt.*;

public final class _p_pm_db extends Panel
```

```
{
    private _p_rs_db    panel = null;
    private _f_rs_db    frame = null;
    private int         controlSize = 8;

    public Rectangle    aheadRect = new Rectangle();    //  current page
    public Rectangle    closeRect = new Rectangle();    //  current page
    public Rectangle    abackRect = new Rectangle();    //  current page

    private final void dbg(String   s)
    {
        if (debug) System.out.println(s);
    }
    private boolean debug = false;                          //  Print debugging info?

    public final Dimension minimumSize()
    {
        String test = "Screen 00 of ~00";
        return new Dimension
            (   controlSize + 4                         //  marker width
            +   (   panel != null
                ?   panel.textMetrics.stringWidth(test)
                :   panel.readerEnWidth * test.length()
                )
            ,   Math.max
                (   panel.footerTextHeight + 1
                ,   controlSize + 4                     //  marker height
                )
            );
    }

    public boolean mouseUp(Event evt, int x, int y)
    {
    //  dbg("_p_pm_db.mouseUp()");

        if (aheadRect.inside(x, y))
        {
            frame.doViewerRight();
            return true;
        }
        else
        if (closeRect.inside(x, y))
        {
            frame.ez_HTML.disposeFrame(frame);
            return true;
        }
        else
        if (abackRect.inside(x, y))
        {
            frame.doViewerLeft();
            return true;
        }
        return false;
    }

    public final void paint(Graphics g)
    {
        Rectangle r = this.bounds();

        if (g != null)
        {
            g.setColor(panel.getBackground());
            g.fillRect(0, 0, r.width, r.height);
            g.draw3DRect(0, 0, r.width - 1, r.height - 1, false);
            g.setColor(panel.getForeground());
            g.setFont(panel.readerFooterFont);
        }

        controlSize = panel.footerTextHeight*2/3;
        setMarkers(g);
        paintMarkers(g);

        StringBuffer pI = new StringBuffer();
        pI.append("Screen ").append(String.valueOf(panel.readerPageNum)).append(" of
");
        if (panel.readerNumPages == 0) pI.append("~");
        pI.append(String.valueOf(panel.readerEstPages));
        String pageInfo = pI.toString();

        int infoWidth = getFontMetrics(panel.readerFooterFont).stringWidth(pageInfo);

        int xLoc = (r.width - infoWidth)/2;
        int yLoc =  r.height - panel.footerTextHeight/5 - 1;
        if (g != null) g.drawString(pageInfo, xLoc, yLoc);
    }

    public final void paintMarkers(Graphics g)
    {
        if (g == null) return;

        Rectangle r = this.bounds();

        g.setColor(panel.getForeground());

        if (panel.readerPageNum > 1)                                  //  left arrow
        {
            int x = abackRect.x + 2
            ,   y = abackRect.y + abackRect.height/2;
            for (int arrow = 0; arrow < controlSize; arrow++)
            {
                g.drawLine(x + arrow,y - arrow/2,x + arrow,y + arrow/2);
            }
        }

        if (panel.readerFrame)                                        //  close box
        {
            int x = closeRect.x + 2
            ,   y = closeRect.y + 2
            ,   w = closeRect.width  - 5
            ,   h = closeRect.height - 5;

            g.drawRect(  x, y,   w,   h);
            g.drawLine(  x, y, x+w, y+h);
            g.drawLine(x+w, y,   x, y+h);
        }

        if (panel.readerPageNum < panel.readerEstPages)               //  right arrow
        {
            int x = aheadRect.x + aheadRect.width - 2
            ,   y = abackRect.y + abackRect.height/2;
            for (int arrow = 0; arrow < controlSize; arrow++)
            {
                g.drawLine(x - arrow,y + arrow/2,x - arrow,y - arrow/2);
            }
        }
    }

    public final Dimension preferredSize()
    {
        return minimumSize();
    }

    public void setMarkers(Graphics g)
    {
        Rectangle r = this.bounds();

        int w = this.size().width;
        int h = this.size().height;

        abackRect.reshape
        (   w - 2 - (controlSize+4)*3
        ,   h - 4 - controlSize
        ,   controlSize + 4
        ,   controlSize + 4
        );

        closeRect.reshape
        (   w - 2 - (controlSize+4)*2
        ,   h - 4 - controlSize
        ,   controlSize + 4
        ,   controlSize + 4
        );

        aheadRect.reshape
        (   w - 2 - (controlSize+4)*1
        ,   h - 4 - controlSize
        ,   controlSize + 4
        ,   controlSize + 4
        );
    }

    public void update(Graphics g)
    {
    //  g.setColor(getBackground());
    //  g.fillRect(0, 0, width, height);
    //  g.setColor(getForeground());

        paint(g);
    }

    _p_pm_db(_p_rs_db panel, _f_rs_db frame)
    {
        this.panel = panel;
        this.frame = frame;
    }
}

/**
 *  %W% %E% Everett Stoub
 *
 *  Copyright (c) '96-'97 ION Web Products, Inc. All Rights Reserved.
 *
 *  an object to store form element data
 *
 *  this class is not downloaded unless needed
 *
 *  @author Everett Stoub %I%, %G%
 */
import java.awt.*;
import java.net.*;
import java.util.*;

public class _o_fe_db extends Object
{
    public final static int NONE        = -1;
    public final static int BUTTON      =  0;
    public final static int CHECKBOX    =  1;
    public final static int FILE        =  2;
    public final static int HIDDEN      =  3;
    public final static int IMAGE       =  4;
    public final static int PASSWORD    =  5;
    public final static int RADIO       =  6;
    public final static int RESET       =  7;
    public final static int SUBMIT      =  8;
    public final static int TEXT        =  9;
    public final static int SELECT      = 10;
    public final static int TEXTAREA    = 11;

//  state variables

```

```
    public int      formNumber; //  ez_HTML formsDB entry number (1,2,...)
    public int      elemNumber; //  the form's element number (1,2,...)
    public int      tIndex;     //  input element type index
    public String   name;       //  i/o:    element name to be passed to the
designated form-processing application

//  twin value state variables: [0] - initial (for reset function) and [1] - current

    public String   value[]     = new String[2];    //  i/o:    value string(s) to be
passed to the designated form-processing application

    public String   accept[]    = new String[2];    //  state:  list of file types for
file input selection
    public boolean  checked[][] = new boolean[2][]; //  state:  selector option or
checkbox or radio selection state: true for passing name & value(s)
    public int      maxlength[] = new int[2];       //  state:  number of accepted
characters to be passed from file, password, and text elements
    public boolean  multiple[]  = new boolean[2];   //  state:  multiple item
selection capability in selector options list

    public URL      src[]       = new URL[2];       //  input:  source url for image

    public int      align[]     = new int[2];       //  view:   index for
_p_rs_db.top, _p_rs_db.middle, or _p_rs_db.bottom image alignment
    public int      border[]    = new int[2];       //  view:   width of image border
inset in pixels
    public int      cols[]      = new int[2];       //  view:   number of displayed
characters across selector panes or textarea fields
    public int      rows[]      = new int[2];       //  view:   number of displayed
rows of textarea
    public int      size[]      = new int[2];       //  view:   number of displayed
characters in file, password, and test fields, or rows of selector panes

    public _o_fe_db(int formNumber, int elemNumber, String tagText)
    {
        dbg("new _o_fe_db("+formNumber+","+elemNumber+",["+tagText+"])");

        this.formNumber = formNumber;
        this.elemNumber = elemNumber;
        this.tIndex = _o_fe_db.getTagAttribTypeInt(tagText);
        this.name = _o_tg_db.getTagAttribString(tagText, "name");

        this.value[0] = this.value[1] = _o_tg_db.getTagAttribString(tagText,
"value");

        this.accept[0] = this.accept[1] = _o_tg_db.getTagAttribString(tagText,
"accept");
    }

//  static methods

    private static final void dbg(String s)
    {
        if (debug) System.out.println(s);
    }
    private static boolean debug = true;    //  Print debugging info?

    public static final int getTagAttribTypeInt(String itsText)
    {
        String itsLabel = _o_tg_db.getTagAttribString(itsText, "type");

        if (itsLabel == null)
        {
            return NONE;
        }

        String lowLabel = itsLabel.toLowerCase();

        if (lowLabel.equals("button"))      return BUTTON;  else
        if (lowLabel.equals("checkbox"))    return CHECKBOX;else
        if (lowLabel.equals("file"))         return FILE;     else
        if (lowLabel.equals("hidden"))       return HIDDEN;   else
        if (lowLabel.equals("image"))        return IMAGE;    else
        if (lowLabel.equals("password"))     return PASSWORD;else
        if (lowLabel.equals("radio"))        return RADIO;    else
        if (lowLabel.equals("reset"))        return RESET;    else
        if (lowLabel.equals("submit"))       return SUBMIT;   else
        if (lowLabel.equals("text"))         return TEXT;     else
        if (lowLabel.equals("select"))       return SELECT;   else
        if (lowLabel.equals("textarea"))     return TEXTAREA;else

        return NONE;
    }
}

/**
 *  %W% %E% Everett Stoub
 *
 *  Copyright (c) '96-'97 ION Web Products, Inc. All Rights Reserved.
 *
 *  an object to present html forms in panels
 *
 *  this class is not downloaded unless needed
 *
 *  @author Everett Stoub %I%, %G%
 */
import java.awt.*;
import java.io.*;
import java.net.*;
import java.util.*;

public class _o_fm_db extends Object
{
    public final static int NONE    = -1;
    public final static int GET     =  0;
    public final static int POST    =  1;

    public final static int STND    =  2;
    public final static int TEXT    =  3;
    public final static int DATA    =  4;

//  state variables

    public int mIndex;                                  //  method index value
    public int eIndex;                                  //  encode index value
    public String action;                               //  target for submit
command

    public int formNumber;                              //  ez_HTML formsDB entry
number (1,2,...)
    public Vector formElements;                         //  the elements of this
form

    public String replaceContent(String formContent)
    {
    //  create private form attribute: the form number

        StringBuffer formsTag = new StringBuffer();
        formsTag.append(_p_rs_db.PF)
                .append(_p_rs_db.FM);                   //  private form tag

        StringBuffer formAttr = new StringBuffer();
        formAttr.append(_p_rs_db.BL)
                .append(_p_rs_db.FN)                    //  private form attribute
                .append(_p_rs_db.EQ)
                .append(formNumber);                    //  form number insertion
string

        String newContent = formContent;
        String LCnContent = newContent.toLowerCase();

    //  create private form element attributes: the form & element sequence numbers

        int elemNumber = 0;
        String insert[] =
        {   _p_rs_db.IN                                        //  input elements
(buttons, etc.)
    //  ,   _p_rs_db.OP                                        //  selector option
elements
        ,   _p_rs_db.ST                                        //  selector elements
        ,   _p_rs_db.TA                                        //  textarea elements
        };
        for (int i = 0; i < insert.length; i++)
        {
            String insertion = _p_rs_db.PF + insert[i];
            int j = LCnContent.indexOf(insertion);             //  the first of this kind
of tag
            int k = LCnContent.indexOf(_p_rs_db.SF, j);        //  its terminus
            while (-1 < j && j < k)
            {
                elemNumber++;

                j += insertion.length();
                k += _p_rs_db.SF.length();                     //  post-tag content range
start point

            //  add this new element to this form

                formElements.addElement
                (   new _o_fe_db
                    (   formNumber
                    ,   elemNumber
                    ,   LCnContent.substring(j,k)              //  the entire element
input tag
                    )
                );

                StringBuffer elemAttr = new StringBuffer();
                elemAttr.append(_p_rs_db.BL)
                        .append(_p_rs_db.EN)
                        .append(_p_rs_db.EQ)
                        .append(elemNumber)                    //  element number
insertion string
                        .append(_p_rs_db.SF);

                StringBuffer text = new StringBuffer();
                text.append(newContent.substring(0,j))         //  preceeding text,
including orig input tag preface
                    .append(formAttr.toString())               //  private form attribute
                    .append(elemAttr.toString())               //  private element
attribute
                    .append(newContent.substring(k));          //  succeeding text,
excluding orig input tag suffix

                newContent = text.toString();
                LCnContent = newContent.toLowerCase();

                j = LCnContent.indexOf(insertion, j);
                k = LCnContent.indexOf(_p_rs_db.SF, j);
            }
        }

        StringBuffer finalTag = new StringBuffer();
        finalTag.append(_p_rs_db.PF)
                .append(_p_rs_db.SL)
                .append(_p_rs_db.FM)                           //  private end form tag
                .append(_p_rs_db.SF);

        StringBuffer output = new StringBuffer();
        output.append(formsTag.toString())
```

```
                .append(formAttr.toString())
                .append(_p_rs_db.SF)
                .append(newContent)
                .append(finalTag.toString());

            return output.toString();
        }

        public int size()
        {
            return formElements.size();
        }

        public _o_fm_db(int formNumber, String formTag)
        {
            dbg("new _o_fm_db("+formNumber+",["+formTag+"])");

            this.mIndex = _o_fm_db.getTagAttribMethodInt(formTag);
            this.eIndex = _o_fm_db.getTagAttribEncodeInt(formTag);
            this.action = _o_tg_db.getTagAttribString(formTag, "action");

            this.formNumber = formNumber;
            formElements = new Vector(10);
        }

//      static methods

        private static final void dbg(String s)
        {
            if (debug) System.out.println(s);
        }
        private static boolean debug = true;     //  Print debugging info?

        public static int addForm
        (   String tagText
        ,   String itsText
        ,   _p_rs_db panel
        ,   int addIndexValue
        ,   Graphics g
        )
        {
            dbg("_o_fm_db.addForm(["+itsText+"],panel,"+addIndexValue+",g)");

            int addIndex = addIndexValue;

            int n = _o_tg_db.getTagAttribInt(tagText, _p_rs_db.FN, 0);
            _o_fm_db form = panel.ez_HTML.getFormDB().getForm(n);
            if (form == null)
            {
                return -addIndex;
            }
            else
            {
                return addIndex;
            }
        }

        public static int addInput
        (   String tagText
        ,   String itsText
        ,   _p_rs_db panel
        ,   int addIndexValue
        ,   Graphics g
        )
        {
            dbg("_o_fm_db.addInput(["+itsText+"],panel,"+addIndexValue+",g)");

            int addIndex = addIndexValue;

            return addIndex;
```

```
    }

    public static int addOption
    (   String tagText
    ,   String itsText
    ,   _p_rs_db panel
    ,   int addIndexValue
    ,   Graphics g
    )
    {
        dbg("_o_fm_db.addOption(["+itsText+"],panel,"+addIndexValue+",g)");

        int addIndex = addIndexValue;

        return addIndex;
    }

    public static int addSelector
    (   String tagText
    ,   String itsText
    ,   _p_rs_db panel
    ,   int addIndexValue
    ,   Graphics g
    )
    {
        dbg("_o_fm_db.addSelector(["+itsText+"],panel,"+addIndexValue+",g)");

        int addIndex = addIndexValue;

        return addIndex;
    }

    public static int addTextArea
    (   String tagText
    ,   String itsText
    ,   _p_rs_db panel
    ,   int addIndexValue
    ,   Graphics g
    )
    {
        dbg("_o_fm_db.addTextArea(["+itsText+"],panel,"+addIndexValue+",g)");

        int addIndex = addIndexValue;

        return addIndex;
    }

    public static int endForm
    (   String tagText
    ,   String itsText
    ,   _p_rs_db panel
    ,   int addIndexValue
    ,   Graphics g
    )
    {
        dbg("_o_fm_db.endForm(["+itsText+"],panel,"+addIndexValue+",g)");

        int addIndex = addIndexValue;

        return addIndex;
    }

    public static int endOption
    (   String tagText
    ,   String itsText
    ,   _p_rs_db panel
    ,   int addIndexValue
    ,   Graphics g
    )
    {
        dbg("_o_fm_db.endOption(["+itsText+"],panel,"+addIndexValue+",g)");

        int addIndex = addIndexValue;

        return addIndex;
    }

    public static int endSelector
    (   String tagText
    ,   String itsText
    ,   _p_rs_db panel
    ,   int addIndexValue
    ,   Graphics g
    )
    {
        dbg("_o_fm_db.endSelector(["+itsText+"],panel,"+addIndexValue+",g)");

        int addIndex = addIndexValue;

        return addIndex;
    }

    public static int endTextArea
    (   String tagText
    ,   String itsText
    ,   _p_rs_db panel
    ,   int addIndexValue
    ,   Graphics g
    )
    {
        dbg("_o_fm_db.endTextArea(["+itsText+"],panel,"+addIndexValue+",g)");

        int addIndex = addIndexValue;

        return addIndex;
    }

    public static final int getTagAttribEncodeInt(String itsText)
    {
        String itsLabel = _o_tg_db.getTagAttribString(itsText, "enctype");

        if (itsLabel == null)
        {
            return NONE;
        }

        String lowLabel = itsLabel.toLowerCase();

        if (lowLabel.equals("application/x-www-form-urlencoded"))   return STND;
    else
        if (lowLabel.equals("text/plain"))                           return TEXT;
    else
        if (lowLabel.equals("multipart/form-data"))                  return DATA;
    else

        return NONE;
    }

    public static final int getTagAttribMethodInt(String itsText)
    {
        String itsLabel = _o_tg_db.getTagAttribString(itsText, "method");

        if (itsLabel == null)
        {
            return NONE;
        }

        String lowLabel = itsLabel.toLowerCase();

        if (lowLabel.equals("get"))      return GET;      else
        if (lowLabel.equals("post"))    return POST;    else

        return NONE;
    }
}

/**
 *  %W% %E% Everett Stoub
 *
 *  Copyright (c) '96-'97 ION Web Products, Inc. All Rights Reserved.
 *
 *  an object to maintain html form objects list
 *
 *  this class is not downloaded unless needed
 *
 *  @author Everett Stoub %I%, %G%
 */
import java.awt.*;
import java.util.*;

public class _o_fl_db extends Object
{
    public Vector formObjects;

    private ez_html ez_HTML;

    public _o_fm_db getForm(int formNumber)
    {
        if (0 < formNumber && formNumber <= formObjects.size())
            return (_o_fm_db)formObjects.elementAt(formNumber-1);
        else
            return null;
    }

    public void newForm(_o_fm_db newForm)
    {
        formObjects.addElement(newForm);
    }

    public _o_fl_db(ez_html ez_HTML)
    {
        formObjects = new Vector(10);

        this.ez_HTML = ez_HTML;
    }
}


/**
 *  %W% %E% Everett Stoub
 *
 *  Copyright (c) '96-'97 ION Web Products, Inc. All Rights Reserved.
 *
 *  a frame for displaying an image, with static methods for handling images
 *
 *  this class is not downloaded unless needed
 *
 *  @author Everett Stoub %I%, %G%
 */
import java.awt.*;
import java.io.*;
import java.net.*;

public class _f_im_db extends Frame
{
    Image itsImage = null;
    private ez_html ez_HTML = null;
    public final static String DI = "Release to display image...";
    public final static String DM = "Drag to move image...";

    int xImg = 0, xDown = 0, xWide = 0;
    int yImg = 0, yDown = 0, yTall = 0;

    private static final void dbg(String s)
    {
        if (debug) System.out.println(s);
    }
    private static boolean debug = false;       //  Print debugging info?

    public final boolean handleEvent(Event evt)
    {
        switch (evt.id)
        {
            case Event.WINDOW_DEICONIFY:    this.show();    break;
            case Event.WINDOW_ICONIFY:      this.hide();    break;
            case Event.WINDOW_DESTROY:      itsImage.flush();
                                            this.dispose(); break;
        }
        return super.handleEvent(evt);
    }

    public final Dimension minimumSize()
    {
        Dimension itsSize = getToolkit().getScreenSize();
        itsSize.width = Math.min(itsSize.width, xWide);
        itsSize.height = Math.min(itsSize.height - 50, yTall); // hack for menus

        return itsSize;
    }

    public boolean mouseDown(Event evt, int x, int y)
    {
    //  dbg("_f_im_db.mouseDown()");

        if (itsImage != null)
            ez_HTML.showStatus(DM);

        xDown = x;
        yDown = y;

        return true;
    }

    public boolean mouseDrag(Event evt, int x, int y)
    {
        if (itsImage != null)
            ez_HTML.showStatus(DI);

        xImg = Math.min(0,xImg + x - xDown);
        yImg = Math.min(0,yImg + y - yDown);

        xDown = x;
        yDown = y;

        repaint();

        return true;
    }

    public boolean mouseEnter(Event evt, int x, int y)
    {
        ez_HTML.showStatus(DM);

        return true;
    }

    public boolean mouseExit(Event evt, int x, int y)
    {
        ez_HTML.showStatus(_p_rs_db.MT);
```

```
            return false;
        }

        public boolean mouseMove(Event evt, int x, int y)
        {
            ez_HTML.showStatus(DM);

            return true;
        }

        public final void paint(Graphics g)
        {
            if (itsImage != null && g != null)
                g.drawImage(itsImage, xImg, yImg, this);
        }

        public final Dimension preferredSize()
        {
            return minimumSize();
        }

        public _f_im_db(String itsTitle, Image itsImage, ez_html ez_HTML)
        {
            this.itsImage = itsImage;
            this.ez_HTML = ez_HTML;

            xWide = itsImage.getWidth(this);
            yTall = itsImage.getHeight(this);

            this.setTitle(itsTitle);
            this.pack();
            this.show();
        }
    }

    /**
     *  %W% %E% Everett Stoub
     *
     *  Copyright (c) '96-'97 ION Web Products, Inc. All Rights Reserved.
     *
     *  an object to load images from files
     *
     *  this class is not downloaded unless needed
     *
     *  @author Everett Stoub %I%, %G%
     */
    import java.awt.*;
    import java.net.*;
    import java.io.*;
    import java.util.*;

    public class _o_im_db extends Object
    {
        private ez_html         ez_HTML         = null;

        private Vector          imageFileNames  = null;        //  vector of all image
    file names for document
        private Vector          graphicsURLs    = null;        //  vector of all graphic
    image URLs for document
        private Vector          graphicsImages  = null;        //  vector of all graphic
    images for document

        public static int placeImage
        (   String tagText
        ,   ez_html ez_HTML
        ,   _p_rs_db panel
        ,   int addIndexValue
        ,   Graphics g
        )
        {
```

```
        String theFileName = _o_tg_db.getTagAttribString(tagText,panel.SR);
        int addIndex = addIndexValue;

        if (theFileName != null)
        {
            Image theImage = ez_HTML.getImageDB().fetchImage(theFileName);

            int theWidth = 0;
            if (theImage != null)
            {
                int theAlignment = _o_tg_db.getTagAlignment(tagText, panel.BOTTOM);
                theWidth = addImage(theImage, theFileName, theAlignment,
panel.tagLength, ez_HTML, panel);
                panel.tagLength = 0;                    //  value saved in buffer
vectors
            }
            if (panel.te + theWidth
            >=  panel.xe - panel.indent(true))          //  it won't fit on this line:
draw pending line
            {
                addIndex += panel.paintRow(g,1);        //  try to punch out pending
line elements
                while (panel.yb > panel.ye)             //  start a new column?
                {
                    if (panel.lastColumn)
                    {
                        return -addIndex;               //  no more columns left to
fill
                    }
                    panel.nextColumn();
                }
                panel.te = panel.xb + panel.indent(false) + theWidth;
            }
            else
            {
                panel.te += theWidth;
            }
            if (panel.te
            ==  panel.xe - panel.indent(true))          //  it fills this line: draw
it out
            {
                addIndex += panel.paintRow(g,0);        //  try to punch out pending
line elements
                while (panel.yb > panel.ye)             //  start a new column?
                {
                    if (panel.lastColumn)
                    {
                        return -addIndex;               //  no more columns left to
fill
                    }
                    panel.nextColumn();
                }
                panel.te = panel.xb + panel.indent(false);
            }
        }
        return addIndex;
    }

    private static final int addImage
    (   Image theImage
    ,   String theFileName
    ,   int theAlignment
    ,   int tagLength
    ,   ez_html ez_HTML
    ,   _p_rs_db panel
    )
    {
        int wIm = theImage.getWidth(ez_HTML);
        int hIm = theImage.getHeight(ez_HTML);
        int wCl = panel.effectiveWidth();                    //  available width of the
column

        if (wIm == 0 || hIm == 0) return 0;                  //  nothing to draw

        if (wIm > wCl)                                       //  scale graphic down to
column width
        {
            hIm = Math.round( (float)wCl * (float)hIm / (float)wIm );
            wIm = wCl;
        }

        if (hIm > panel.readerRect.height)                   //  scale graphic down to
column height
        {
            wIm = Math.round( (float)panel.readerRect.height * (float)wIm / (float)hIm
);
            hIm = panel.readerRect.height;
        }

        Image    anOffImage = ez_HTML.createImage(wIm,hIm);
        Graphics itsGraphic = anOffImage.getGraphics();

    //  itsGraphic.setColor(panel.getBackground());
    //  itsGraphic.fillRect(0, 0, wIm, hIm);
        itsGraphic.drawImage(theImage, 0, 0, wIm, hIm, panel);
        itsGraphic.dispose();

        int maxAscent = hIm;                                 //  default case (BOTTOM)
        if (theAlignment == panel.MIDDLE)
            maxAscent = hIm/2;  else
        if (theAlignment == panel.TOP)
            maxAscent = panel.textHalfSpace*3/2;             //  cases LEFT & RIGHT are
not handled yet

        panel.lineImages.addElement(anOffImage);
        panel.dataValues.addElement
        (   new Rectangle
            (   tagLength                                    //  "x"        increment
for endIndex (character count)
            ,   maxAscent                                    //  "y"        for
vertical placement of image (ascent) above baseline
            ,   wIm                                          //  "width"    element
width on line
            ,   hIm - maxAscent                              //  "height"   element
drop below baseline (descent)
            )
        );
        panel.wordBlocks.addElement(theFileName);            //  image cross reference
        panel.hyperLinks.addElement(panel.hyperHead);
        panel.styleCodes.addElement(panel.styleCode());      //  ignored in rendering
images
        panel.wordColors.addElement(panel.textColor);
        panel.charsFonts.addElement(panel.textFont);

        return wIm;                                          //  the width of this
element on the line
    }

    public Image fetchImage(String fileName)
    {
        if (fileName == null)
        {
            return null;
        }
        else
        {
            Image theImage = null;

            int imageDatabaseIndex =                    //  get reference to image
database file name
                imageFileNames.indexOf(fileName);

            if (imageDatabaseIndex < 0)                 //  not registered yet:
add it in
            {
                URL theURL = null;

                try                                     //  look in the
documentBase
                {
                    theURL = new URL(ez_HTML.getDocumentBase(), fileName);
                    InputStream is = theURL.openStream();   //  test connection
                    theImage = ez_HTML.getImage(theURL);
                    ez_HTML.getTracker().addImage(theImage,0);

                //  jump-start image loading

                    boolean status = ez_HTML.getTracker().checkID(0,true);
                    try
                    {
                        ez_HTML.getTracker().waitForID(0);
                    }
                    catch ( InterruptedException ie ) { }
                    Image bufferedImage = ez_HTML.createImage(1,1);
                    if (bufferedImage != null)
                    {
                        Graphics bg = bufferedImage.getGraphics();
                        if (bg != null)
                        {
                            bg.drawImage(theImage,0,0,this.ez_HTML);
                            bg.dispose();
                        }
                    }
                    bufferedImage.flush();
                }
                catch (MalformedURLException u)
                {
                    System.out.println("_o_im_db.fetchImage("+fileName+"):
MalformedURLException = "+u);
                    theImage = null;
                }
                catch (IOException io)
                {
                    System.out.println("_o_im_db.fetchImage("+fileName+"): InputStream
IOException = "+io);
                    theImage = null;
                }

            //  update image database for document

                imageFileNames.addElement(fileName);        //  the filename is the
main key to image database access
                graphicsURLs.addElement(theURL);            //  in case we need to
download it again later
                graphicsImages.addElement(theImage);        //  this may be reset to
null with imageUpdate
            }
            else
            {
                theImage = (Image)graphicsImages.elementAt(imageDatabaseIndex);
            }
            return theImage;
        }

    }

    public static final void zoomImage                      //  new frame with full-
size rendering
```

```
    (   String theFileName
    ,   ez_html ez_HTML
    )
    {
        Image theImage = ez_HTML.getImageDB().fetchImage(theFileName);

        if (theImage != null)
        {
            _f_im_db itsFrame = new _f_im_db(theFileName, theImage, ez_HTML);
            itsFrame.setBackground(ez_HTML.getBackground());
            itsFrame.setForeground(ez_HTML.getForeground());
        }
    }

    public _o_im_db(ez_html ez_HTML)
    {
        this.ez_HTML = ez_HTML;

        imageFileNames  = new Vector(10);                    //  must be created before
calls to fetchImage
        graphicsURLs    = new Vector(10);                    //  must be created before
calls to fetchImage
        graphicsImages  = new Vector(10);                    //  must be created before
calls to fetchImage
    }
}

/**
 *  %W% %E% Everett Stoub
 *
 *  Copyright (c) '96-'97 ION Web Products, Inc. All Rights Reserved.
 *
 *  a frame for displaying a table
 *
 *  this class is not downloaded unless needed
 *
 *  @author Everett Stoub %I%, %G%
 */
import java.awt.*;
import java.io.*;
import java.net.*;

public class _f_tb_db extends Frame
{
    private boolean fullSize = true;            //  appearance switch

    public final static String DT = "Drag to move table...";
    public final static String RT = "Release to view table...";
    public final static String WT = "Wait for new table view...";

    private _p_tb_db itsTable = null;
    private ez_html ez_HTML = null;

    private int xTbl = 0, xDown = 0;
    private int yTbl = 0, yDown = 0;

    public boolean justDragging = false;

    private static final void dbg(String s)
    {
        if (debug) System.out.println(s);
    }
    private static boolean debug = false;       //  Print debugging info?

    public final boolean handleEvent(Event evt)
    {
        switch (evt.id)
        {
            case Event.WINDOW_DEICONIFY:    this.show();    break;
            case Event.WINDOW_ICONIFY:      this.hide();    break;
            case Event.WINDOW_DESTROY:      this.dispose(); break;
        }
        return super.handleEvent(evt);
    }

    public final Dimension minimumSize()
    {
        Dimension s = getToolkit().getScreenSize();
        if (debug)
        {
            s.width  /= 2;
            s.height /= 2;
        }
        else
        {
            s.height -= 50; // hack for menus
        }

        Dimension t = itsTable.minimumSize();

        s.width  = Math.min(s.width,  t.width );
        s.height = Math.min(s.height, t.height);

    //  dbg("_f_tb_db.minimumSize(): screen
"+getToolkit().getScreenSize().width+"x"+getToolkit().getScreenSize().height+", min
table "+t.width+"x"+t.height+", min frame "+s.width+"x"+s.height);

        return s;
    }

    public boolean mouseDown(Event evt, int x, int y)
    {
    //  dbg("_f_tb_db.mouseDown("+x+","+y+")");

        if (itsTable != null)
            ez_HTML.showStatus(DT);

        xTbl = itsTable.location().x;
        yTbl = itsTable.location().y;

        xDown = x;
        yDown = y;

        justDragging = false;

        return true;
    }

    public boolean mouseDrag(Event evt, int x, int y)
    {
        xTbl += x - xDown;
        yTbl += y - yDown;

        xDown = x;
        yDown = y;

        if (itsTable != null)
        {
            ez_HTML.showStatus(RT);

            itsTable.move(xTbl, yTbl);
        }
        justDragging = true;
        repaint();

        return true;
    }

    public boolean mouseEnter(Event evt, int x, int y)
    {
        ez_HTML.showStatus(DT);

        return true;
    }

    public boolean mouseExit(Event evt, int x, int y)
    {
        ez_HTML.showStatus(_p_rs_db.MT);

        return false;
    }

    public boolean mouseMove(Event evt, int x, int y)
    {
        ez_HTML.showStatus(DT);

        return true;
    }

    public boolean mouseUp(Event evt, int x, int y)
    {
    //  dbg("_f_tb_db.mouseUp()");

        if (justDragging)
        {
            ez_HTML.showStatus(WT);
            repaint();
            justDragging = false;
            return true;
        }

        return true;
    }

    public final Dimension preferredSize()
    {
        Dimension s = getToolkit().getScreenSize();
        if (debug)
        {
            s.width  /= 2;
            s.height /= 2;
        }
        else
        {
            s.height -= 50; // hack for menus
        }

        Dimension t = itsTable.preferredSize();

        s.width  = Math.min(s.width,  t.width  * (fullSize?5:4)/4);
        s.height = Math.min(s.height, t.height * (fullSize?5:4)/4);

    //  dbg("_f_tb_db.preferredSize(): screen
"+getToolkit().getScreenSize().width+"x"+getToolkit().getScreenSize().height+", pref
table "+t.width+"x"+t.height+", pref frame "+s.width+"x"+s.height);

        return s;
    }

    public _f_tb_db(String itsTitle, _p_tb_db itsTable, _p_rs_db panel)
    {
    //  dbg("new _f_tb_db("+itsTitle+", "+itsTable.theTableNumber+")");

        this.ez_HTML = panel.ez_HTML;
        this.itsTable = itsTable;
        itsTable.setFontSizeIndex(panel.readerSizeIndex);

        this.setTitle(itsTitle);
        this.setBackground(panel.getBackground());
        this.setForeground(panel.getForeground());
        this.setLayout(new BorderLayout());

        Panel p = new Panel();              //  help prevent WinOS header/footer
coverup
        this.add("Center", p);

        if (fullSize)                       //  want overlarge tables to stretch
beyond window?
        {
            p.add(itsTable);                //  default flow layout
        }
        else                                //  or overlarge tables to shrink to
window area
        {
            p.setLayout(new BorderLayout());
            p.add("Center", itsTable);
        }

        this.pack();
        this.show();

        xTbl = itsTable.location().x;
        yTbl = itsTable.location().y;
    }
}

/**
 *  %W% %E% Everett Stoub
 *
 *  Copyright (c) '96-'97 ION Web Products, Inc. All Rights Reserved.
 *
 *  an object to load tables from specifications
 *
 *  this class is not downloaded unless needed
 *
 *  @author Everett Stoub %I%, %G%
 */
import java.awt.*;
import java.net.*;
import java.io.*;
import java.util.*;

public class _o_tb_db extends Object
{
    private ez_html ez_HTML = null;
    private Vector tableObjects = null;             //  vector of all table objects
for document

    public void addTable(_p_tb_db newTable)
    {
        tableObjects.addElement(newTable);
    }

    public _p_tb_db getCopy(int number, boolean readerFrame)
    {
        _p_tb_db orig = getTable(number);

        if (orig == null)
        {
            return null;
        }
        else
        {
            _p_tb_db aCopy = orig.copy();
            aCopy.readerFrame = readerFrame;
            return aCopy;
        }
    }

    private _p_tb_db getTable(int number)
    {
        if (number < 1 || number > tableObjects.size())
        {
            return null;
        }
        else
        {
            return (_p_tb_db)tableObjects.elementAt(number - 1);
        }
    }

    public int size()
    {
        return tableObjects.size();
    }

    public _o_tb_db(ez_html ez_HTML)
    {
        this.ez_HTML = ez_HTML;
        tableObjects = new Vector(10);
    }

//  static methods

    private static final void dbg(String s)
    {
        if (debug) System.out.println(s);
    }
    private static boolean debug = false;    //  Print debugging info?

    public static int placeTable
    (   String tagText
    ,   ez_html ez_HTML
    ,   _p_rs_db panel
    ,   int addIndexValue
    ,   Graphics g
    )
    {
        int addIndex = addIndexValue;
        int tableNumber = _o_tg_db.getTagAttribInt(tagText, _p_rs_db.VL, 0);

    //  dbg("_o_tb_db.placeTable("+tableNumber+"):
panel("+panel.readerContent.substring(0,Math.min(30,panel.readerContent.length()))+(pa
nel.readerContent.length()<30?panel.ZZ.substring(panel.readerContent.length()):panel.M
T)+")");

        if (tableNumber > 0)
        {
            _p_tb_db theTable = ez_HTML.getTableDB().getCopy(tableNumber,
panel.readerFrame);

            if (theTable != null)
            {
                addIndex += panel.paintRow(g,0);         //  try to punch out pending
line elements
                theTable.setBackground(panel.getBackground());
                theTable.setForeground(panel.getForeground());

                panel.add(theTable);                     //  reshaping below
                panel.pageComps.addElement(theTable);    //  for playback
                theTable.setFontSizeIndex(panel.readerSizeIndex - 1);

                Dimension c = panel.columnSize();        //  as big as it can be
                Dimension p = theTable.preferredSize();  //  as good as it can be
                Dimension m = theTable.minimumSize();    //  as small as it can be
                Dimension t = new Dimension(c);          //  new table dimensions:
default to column
```

```
            //  dbg("_o_tb_db.placeTable("+tableNumber+"): panel col
"+c.width+"x"+c.height+", pref table "+p.width+"x"+p.height+", min table
"+m.width+"x"+m.height);

                if (p.width < c.width)                  //  reduce width to preferred
width?
                    t.width = p.width;                  //  preferred width OK

                if (p.height < c.height)                //  reduce height to preferred
height?
                    t.height = p.height;                //  preferred height OK

                int clearH = panel.ye - panel.yb;       //  headroom in column for
table
                if (clearH < t.height)                  //  table height too tall?
                {
                    if                                  //  room for nice first row?
                    (   theTable.prefRowHeight(0) < clearH
                    ||  m.height < clearH               //  or room for all minimum
rows?
                    ||  10*clearH > c.height            //  or a full column won't
help (much)
                    )
                    {
                        t.height = clearH;              //  at least one row fits into
the headroom
                    }
                    else                                //  this is the last column:
try harder
                    if (!panel.lastColumn)              //  just start a new column
                    {
                        panel.nextColumn();
                    }
                    else                                //  it'll have to wait for the
sequel
                    {
                        panel.remove(theTable);         //  drop it
                        panel.pageComps.removeElementAt(panel.pageComps.size() - 1);
                        return -addIndex;               //  no more columns left to
fill
                    }
                }
            //  dbg("_o_tb_db.placeTable(): reshape table to "+t.width+"x"+t.height);
                theTable.reshape
                (   panel.xb + Math.max(0, (panel.effectiveWidth() - t.width)) / 2
                ,   panel.yb
                ,   t.width
                ,   t.height
                );
                theTable.show();

                panel.lineImages.addElement(null);
                panel.dataValues.addElement
                (   new Rectangle
                    (   panel.tagLength                 //  "x"         increment for
endIndex (character count)
                    ,   0                               //  "y"         for vertical
placement of image (ascent) above baseline
                    ,   t.width                         //  "width"     element width
on line
                    ,   t.height                        //  "height"    element drop
below baseline (descent)
                    )
                );
                panel.wordBlocks.addElement(theTable);  //  table reference
                panel.hyperLinks.addElement(panel.hyperHead);
                panel.styleCodes.addElement(panel.styleCode());
                panel.wordColors.addElement(panel.textColor);
                panel.charsFonts.addElement(panel.textFont);

                panel.tagLength = 0;                    //  value saved in buffer
vectors
            }
        }
        return addIndex;
    }

    public static final void zoomTable                  //  new frame with full-size
rendering
    (   Integer theTableNumber
    ,   _p_rs_db panel
    )
    {
    //  dbg("_o_tb_db.zoomTable("+theTableNumber+","+ez_HTML+")");

        int tableNumber = theTableNumber.intValue();
        if (tableNumber <= panel.ez_HTML.getTableDB().size())
        {
            _p_tb_db theTable = panel.ez_HTML.getTableDB().getCopy(tableNumber, true);
            String itsTitle =
            (   theTable.theCaption == null
            ?   "Table "+tableNumber
            :   theTable.theCaption
            );
            _f_tb_db itsFrame = new _f_tb_db(itsTitle, theTable, panel);
        }
    }
}

/**
 *  %W% %E% Everett Stoub
 *
 *  Copyright (c) '96-'97 ION Web Products, Inc. All Rights Reserved.
 *
 *  an object to store table row data
 *
 *  this class is not downloaded unless needed
 *
 *  @author Everett Stoub %I%, %G%
 */
import java.awt.*;
import java.util.*;

public class _o_tr_db extends Object
{
    public Color    rowBg   = null;
    public int      align   = _p_rs_db.LEFT;
    public int      vAlign  = _p_rs_db.CENTER;

    public Vector   itsCells = new Vector();

    public _o_tr_db
    (   Color rowBg
    ,   int align
    ,   int vAlign
    )
    {
        this.rowBg = rowBg;
        this.align = align;
        this.vAlign = vAlign;
    }
}

/**
 *  %W% %E% Everett Stoub
 *
 *  Copyright (c) '96-'97 ION Web Products, Inc. All Rights Reserved.
 *
 *  an object to handle html tables
 *
 * this class is not downloaded unless needed
 *
 * HTML tag parsing honors the following commands and attributes
 *
 *   • <table>...</table>            standard new table tag
 *         align=type                    left, center, right
 *         bgcolor=color                 rgb or color name
 *         border=pixels                 cell embossed border size
 *         cellpadding=pixels            cell inset size
 *         cellspacing=pixels            border embossement size
 *         hspace=pixels                 table width inset
 *         valign=type                   top, bottom
 *         vspace=pixels                 table height inset
 *         width=pixels or %             table rel or abs width preference
 *
 *   • <tr>...</tr>                  standard new row tag
 *         align=type                    left, center, right
 *         bgcolor=color                 rgb or color name
 *         valign=type                   top, bottom
 *
 *   • <td>...</td>                  standard new column data tag
 *   • <th>...</th>                  standard new column header tag
 *         align=type                    left, center, right
 *         bgcolor=color                 rgb or color name
 *         valign=type                   top, center, bottom
 *
 *   • <caption>...</caption>        standard new row tag
 *         align=type                    top, bottom
 *
 * @author Everett Stoub %I%, %G%
 */
import java.awt.*;
import java.awt.image.*;
import java.util.*;

public class _p_tb_db extends Panel
{
    public final static String  BC          = "bgcolor";
    public final static String  BO          = "border";
    public final static String  CP          = "cellpadding";
    public final static String  CS          = "cellspacing";
    public final static String  HT          = "height";
    public final static String  SH          = "hspace";
    public final static String  SV          = "vspace";
    public final static String  TA          = "table";
    public final static String  TC          = "caption";
    public final static String  TD          = "td";
    public final static String  TH          = "th";
    public final static String  TR          = "tr";
    public final static String  VN          = "valign";
    public final static String  ZM          = "Click to zoom table...";
    public final static String  ZR          = "Release for fullsize table view...";
    public final static String  ZW          = "Wait for fullsize table view...";

    public ez_html ez_HTML = null;
    public Integer theTableNumber = new Integer(0);
    public int fontSizeIndex = 2;
    public boolean readerFrame = false;
    public boolean tableSpecParsed = false;
    public boolean layoutComplete = false;

    private String tableSpec = null;

//  table parameters

    private int         align = _p_rs_db.LEFT;       //  current setting
    private int         vAlign  = _p_rs_db.CENTER;
    private int         hSpace = 0;                   //  current setting
    private int         vSpace = 0;                   //  current setting
    private int         border = 0;                   //  current setting
    private int         cellPadding = 0;            //  current setting
    private int         cellSpacing = 0;            //  current setting
    private int         minWidth = 0;               //  current guideline for whole
table
    private int         minHeight = 0;              //  current guideline for whole
table

    private Insets      insets = new Insets(0,0,0,0);
    private Vector      tableRows = new Vector();   //  each entry is a row object
    private _p_tc_db    tableCells[][];             //  array of _p_tc_db cell objects
    private Dimension   tableSize = new Dimension();
    private int         nRows = 1;
    private int         nCols = 1;
    private int         itsWidth = 0;
    private int         itsHeight = 0;

    public String       theCaption = null;
    private int         capAlign = _p_rs_db.TOP;    //  current setting

    public String addCaption(boolean upperCaption)
    {
        if
        (   ( theCaption == null)
        ||  ( upperCaption && capAlign != _p_rs_db.TOP)
        ||  (!upperCaption && capAlign == _p_rs_db.TOP)
        )
        return _p_rs_db.MT;

        StringBuffer itsContent = new StringBuffer();

        itsContent                      //  new break, with center formatting
            .append(_p_rs_db.PF)
                .append(_p_rs_db.BR).append(_p_rs_db.BL)
                .append(_p_rs_db.AL).append(_p_rs_db.EQ)
                .append(_p_rs_db.CJ)
            .append(_p_rs_db.SF);
        itsContent.append(theCaption);
        itsContent                      //  end paragraph
            .append(_p_rs_db.PF)
                .append(_p_rs_db.SL)
                .append(_p_rs_db.PA)
            .append(_p_rs_db.SF);

    //  dbg("_p_tb_db("+theTableNumber+").addCaption("+upperCaption+"):
"+itsContent.toString());

        return itsContent.toString();
    }

    private void addContent(String content)
    {
    //  dbg("_p_tb_db("+theTableNumber+").addContent("+content+")");

        if (theCaption == _p_rs_db.MT)          //  load content as the caption
        {
            if (content == _p_rs_db.MT)         //  preclude regression
            {
                theCaption = null;              //  complete the load
            }
            else
            {
            //  dbg("... theCaption = "+content);
                theCaption = content;           //  capture the load
            }
        }
        else
        if (tableRows.size() > 0)               //  content could be extraneous
        {
            _o_tr_db thisRow = (_o_tr_db)tableRows.lastElement();

            if (thisRow.itsCells.size() > 0)
            {   _p_tc_db thisCell = (_p_tc_db)thisRow.itsCells.lastElement();

            //  dbg("... the cell(R"+tableRows.size()+"C"+thisRow.itsCells.size()+") =
"+content);
                thisCell.content += content;    //  capture the load
            }
        }
    }

    private void addElement(String itsText)
    {
    //  dbg("_p_tb_db("+theTableNumber+").addElement("+itsText+")");

        String tagText = itsText.toLowerCase();

        if (tagText.startsWith(TA))                                         //  an
html new table tag w/ attributes
        {
            setBackground
            (   _o_tg_db.parseColorValue
                (   _o_tg_db.getTagAttribString(tagText, BC)
                ,   getBackground().getRGB()
                )
            );
            align = _o_tg_db.getTagAlignment(tagText, align);
            vAlign = _o_tg_db.getTagAlignment(tagText, VN, vAlign);

            hSpace = _o_tg_db.getTagAttribInt(tagText,SH,hSpace);
            vSpace = _o_tg_db.getTagAttribInt(tagText,SV,vSpace);

            minWidth = _o_tg_db.getTagPerCentWidth(tagText);                //  coded
negative for per cent value, positive for pixel value
            minHeight = _o_tg_db.getTagAttribInt(tagText,HT,minHeight);

            if (tagText.indexOf(BO) > -1)
                border = _o_tg_db.getTagAttribInt(tagText,BO, 1);
            cellPadding = _o_tg_db.getTagAttribInt(tagText,CP,cellPadding);
            cellSpacing = _o_tg_db.getTagAttribInt(tagText,CS,cellSpacing);
        }
        else if (tagText.startsWith(TR))                                    //  an
html new row tag w/ attributes
        {
        //  dbg("... new table row: tagText = "+tagText);
            _o_tr_db thisRow = new _o_tr_db
            (   _o_tg_db.parseColorValue
                (   _o_tg_db.getTagAttribString(tagText, BC)
                ,   getBackground().getRGB()
                )
            ,   _o_tg_db.getTagAlignment(tagText, align)
            ,   _o_tg_db.getTagAlignment(tagText, VN, vAlign)
            );
            tableRows.addElement(thisRow);                                  //  align,
bgcolor, valign, cells vector
            nRows = tableRows.size();
        }
        else if (tagText.startsWith(TH)                                     //  an
html column header tag w/ attributes for current row
             ||  tagText.startsWith(TD))                                    //  an
html column data tag w/ attributes for current row
        {
            _o_tr_db thisRow = (_o_tr_db)tableRows.lastElement();

            _p_tc_db thisCell = new _p_tc_db                                //
    background color
            (   _o_tg_db.parseColorValue
                (   _o_tg_db.getTagAttribString
                    (   tagText, BC)
                    ,   thisRow.rowBg.getRGB()
                    )
                ,   getForeground()
                ,   _o_tg_db.getTagAlignment(tagText, thisRow.align)          //
    horizontal alignment
                ,   _o_tg_db.getTagAlignment(tagText,VN,thisRow.vAlign)       //
    vertical alignment
                ,   _o_tg_db.getTagPerCentWidth(tagText)                      //
    minimum cell width
                ,   _o_tg_db.getTagAttribInt(tagText,"colspan",1)             //  column
group value
                ,   _o_tg_db.getTagAttribInt(tagText,"rowspan",1)             //  row
group value
                ,   _o_tg_db.getTagAttribInt(tagText,HT,23)                   //
    minimum cell height
                ,   border
                ,   cellPadding
                ,   cellSpacing
                ,   tagText.startsWith(TD)                                    //
    boolean isData
                );
            thisRow.itsCells.addElement(thisCell);
            int rowCells = 0;
            for (int i = 0; i < thisRow.itsCells.size(); i++)
                rowCells += ((_p_tc_db)thisRow.itsCells.elementAt(i)).colSpan;
            nCols = Math.max(nCols, rowCells);
        }
        else if (tagText.startsWith(_p_rs_db.TB))                             //  a
private table in a cell tag
        {
        //  dbg("... finding row "+tableRows.size()+" object");
            _o_tr_db thisRow = (_o_tr_db)tableRows.lastElement();
        //  dbg("... finding cell "+thisRow.itsCells.size()+" object");
            _p_tc_db thisCell = (_p_tc_db)thisRow.itsCells.lastElement();

            thisCell.content += "<"+tagText+">";                              //
    capture the load
        }
        else if (tagText.startsWith(TC))                                      //  an
html caption tag w/ attributes
        {
            theCaption = _p_rs_db.MT;                                         //  code:
next content value is caption
            capAlign = _o_tg_db.getTagAlignment(tagText, capAlign);
        }
        else
        if
        (   tagText.startsWith(_p_rs_db.SL+TA)                                //  an
html end new table tag w/ attributes
        ||  tagText.startsWith(_p_rs_db.SL+TR)                                //  an
html end new row tag w/ attributes
        ||  tagText.startsWith(_p_rs_db.SL+TH)                                //  an
html end column header tag w/ attributes for current row
        ||  tagText.startsWith(_p_rs_db.SL+TD)                                //  an
html end column data tag w/ attributes for current row
        ||  tagText.startsWith(_p_rs_db.SL+_p_rs_db.TB)                       //  a
private end table in a cell tag
        ||  tagText.startsWith(_p_rs_db.SL+TC)                                //  an
html end caption tag w/ attributes
        )
        {
        //  just ignore these
        }
        else
        {
        //  dbg("... unhandled tagText = <"+tagText+">");

            _o_tr_db thisRow = (_o_tr_db)tableRows.lastElement();
            _p_tc_db thisCell = (_p_tc_db)thisRow.itsCells.lastElement();

            thisCell.content += "<"+tagText+">";                              //
    capture the load
```

```
          }
      }

      public _p_tb_db copy()
      {
      //  dbg("_p_tb_db("+theTableNumber+").copy()");

          _p_tb_db aCopy = new _p_tb_db
          (   this.tableSpec
          ,   this.theTableNumber
          ,   this.ez_HTML
          );

          return aCopy;
      }

      private final void dbg(String s)
      {
          if (debug) System.out.println(s);
      }
      private boolean debug = false;              //  Print debugging info?

      public synchronized void doLayout()
      {
      //  dbg("_p_tb_db("+theTableNumber+").doLayout(): "+nRows+"x"+nCols+" table");

          if (!tableSpecParsed)
              parseTableSpec();

          boolean zoomableCells = false;          //  true if top level is _f_tb_db
          Component ancestor = getParent();
          while (ancestor != null)
          {
              if (ancestor instanceof _f_tb_db)
                  zoomableCells = true;
              ancestor = ancestor.getParent();
          }

          tableCells = new _p_tc_db[nRows][nCols];
          for (int j = 0; j < nRows; j++)         //  assign row & column #'s to each
cell
          {
              _o_tr_db thisRow = (_o_tr_db)tableRows.elementAt(j);
              int nCells = thisRow.itsCells.size();
              for (int i = 0; i < nCells; i++)    //  locate all cells in this row
              {
                  _p_tc_db thisCell = (_p_tc_db)thisRow.itsCells.elementAt(i);
                  StringBuffer itsContent = new StringBuffer();
                  if (thisCell.content.length() > 0)
                  {
                      itsContent.append(_p_rs_db.PF).append(_p_rs_db.PA);
                      if (!thisCell.isData        //  center formatting
                      || thisCell.align == _p_rs_db.CENTER)

    itsContent.append(_p_rs_db.BL).append(_p_rs_db.AL).append(_p_rs_db.EQ).append(_p_r
s_db.CJ);
                      itsContent.append(_p_rs_db.SF);

                      if (!thisCell.isData)       //  bold formatting

    itsContent.append(_p_rs_db.PF).append(_p_rs_db.BD).append(_p_rs_db.SF).append(this
Cell.content).append(_p_rs_db.PF).append(_p_rs_db.SL).append(_p_rs_db.BD).append(_p_rs
_db.SF);
                      else                        //  normal formatting
                          itsContent.append(thisCell.content);



    itsContent.append(_p_rs_db.PF).append(_p_rs_db.SL).append(_p_rs_db.PA).append(_p_r
s_db.SF);
                  }
```

```
                _p_rs_db aPane = new _p_rs_db   //  a panel component
                (   itsContent.toString()       //  specified html content
                ,   ez_HTML                     //  the applet
                ,   null                        //  no frame
                ,   false                       //  NOT the reader screen
                ,   1                           //  maxColumns
                ,   false                       //  no paging indicator marks
                ,   false                       //  no page position footer
                ,   thisCell.cellBg
                ,   thisCell.cellFg
                ,   (itsContent.length() > 0 ? border: 0)
                ,   cellPadding
                ,   cellSpacing
                ,   zoomableCells               //  zoomability
                );
                aPane.setFontSizeIndex(this.fontSizeIndex);

                int c = i, r = j;
                while (tableCells[r][c] != null)//  find first empty table cell (r,c)
                if (++c == nCols)               //  start a new row!
                {
                    c = 0;                      //  start new row with 1st column
                    if (++r == nRows)           //  no more rows: attempt recovery
                    {
                        c = i;  r = j;
                        break;                  //  could replace prior cell
                    }
                }
                if (tableCells[r][c] == null)   //  prevent cell replacement
                {
                    thisCell.colNum = c;    thisCell.rowNum = r;
                    for (int f = 0; f < thisCell.rowSpan && r + f < nRows; f++)
                        for (int g = 0; g < thisCell.colSpan && c + g < nCols; g++)
                            tableCells[r + f][c + g]
                                = thisCell;     //  fill the element's footprint
                }
                this.add(thisCell);             //  add the cell to this table panel
(reshape later)
                thisCell.add("Center", aPane);  //  add the reader panel to the cell
                thisCell.show();
            }
        }
        layoutComplete = true;
        this.reshapeAll();
        this.invalidate();
        this.validate();
    }

    public Insets insets()
    {
        return insets;
    }

    private int minColLoc(int c)
    {
        int x = insets.left;
        for (int i = 0; i < c; i++)
            x += minColWidth(i);

        return x;
    }

    private int minColLoc(_p_tc_db c)
    {
        return minColLoc(c.colNum);
    }

    private int minColWidth(int c)
    {
        int minColWidth = 0;
```

```
            for (int r = 0; r < nRows; r++)
                if (tableCells[r][c] != null)
                    minColWidth = Math.max
                    (   minColWidth
                    ,   tableCells[r][c].minimumSize().width
                    /   tableCells[r][c].colSpan
                    );

            return minColWidth;
        }

        private int minColWidth(_p_tc_db c)
        {
            return minColWidth(c.colNum);
        }

        public Dimension minimumSize()
        {
            if (!layoutComplete)
                doLayout();

        //
        dbg("_p_tb_db("+theTableNumber+").minimumSize("+minTableSize().width+","+minTableS
ize().height+")");

            return minTableSize();
        }

        public int minRowHeight(int r)
        {
            int minRowHeight = 0;
            for (int c = 0; c < nCols; c++)
                if (tableCells[r][c] != null)
                    minRowHeight = Math.max
                    (   minRowHeight
                    ,   tableCells[r][c].minimumSize().height
                    /   tableCells[r][c].rowSpan
                    );

            return minRowHeight;
        }

        private int minRowHeight(_p_tc_db c)
        {
            return minRowHeight(c.rowNum);
        }

        private int minRowLoc(int r)
        {
            int y = insets.top;
            for (int j = 0; j < r; j++)
                y += minRowHeight(j);

            return y;
        }

        private int minRowLoc(_p_tc_db c)
        {
            return minRowLoc(c.rowNum);
        }

        public Dimension minTableSize()
        {
            Dimension d = new Dimension();

            for (int c = 0; c < nCols; c++)
                d.width += minColWidth(c);

            for (int r = 0; r < nRows; r++)
                d.height += minRowHeight(r);

        d.width += insets.left + insets.right;
        d.height += insets.top + insets.bottom;

        return d;
    }

    public boolean mouseDown(Event evt, int x, int y)
    {
    //  dbg("_p_tb_db.mouseDown("+x+","+y+")");

        if (getParent() != null)
        {
            if (getParent() instanceof _p_rs_db)
            {
                ez_HTML.showStatus(ZR);
                return true;
            }
            return false;
        }

        return true;
    }

    public boolean mouseEnter(Event evt, int x, int y)
    {
        if (getParent() instanceof _p_rs_db)
            ez_HTML.showStatus(ZM);

        return true;
    }

    public boolean mouseMove(Event evt, int x, int y)
    {
        if (getParent() instanceof _p_rs_db)
            ez_HTML.showStatus(ZM);

        return true;
    }

    public boolean mouseExit(Event evt, int x, int y)
    {
        ez_HTML.showStatus(_p_rs_db.MT);

        return true;
    }

    public boolean mouseUp(Event evt, int x, int y)
    {
    //  dbg("_p_tb_db.mouseUp()");

        if (getParent() != null)
        {
            if (getParent() instanceof _p_rs_db)
            {
            //  dbg("_p_tb_db.mouseUp(): zooming table "+this.theTableNumber);

                ez_HTML.showStatus(ZW);
                _o_tb_db.zoomTable
                (   this.theTableNumber
                ,   (_p_rs_db)getParent()
                );
                return true;
            }
            else
            {
                return false;
            }
        }

        return super.mouseUp(evt, x, y);
    }

    public synchronized void paint(Graphics g)
    {
    //  dbg("_p_tb_db("+theTableNumber+", "+countComponents()+" components:
"+size().width+" x "+size().height+").paint(g): isEnabled = "+isEnabled()+", isShowing
= "+isShowing()+", isValid = "+isValid()+", isVisible = "+isVisible()+",
layoutComplete = "+layoutComplete);

        if (!layoutComplete)
            doLayout();

        Rectangle r = bounds();

        if (g != null)
        {
            g.setColor(getBackground());
            g.fillRect(0, 0, r.width, r.height);
            for (int i = 0; i < border; i++)
                g.draw3DRect
                (   i + hSpace + cellSpacing/2
                ,   i + vSpace + cellSpacing/2
                ,   r.width  - 1 - 2*hSpace - cellSpacing - 2*i
                ,   r.height - 1 - 2*vSpace - cellSpacing - 2*i
                ,   true
                );
            g.setColor(getForeground());
        }
    }

    private void parseTableSpec()
    {
        StringTokenizer tagBlocks = new StringTokenizer         //  break up the text
into normal tagBlocks
            (tableSpec,_p_rs_db.PF,true);                       //  final true returns
token on next call

        while (tagBlocks.hasMoreTokens())                       //  tag block to
follow
        {
            String tagBlock = tagBlocks.nextToken(_p_rs_db.PF); //  [<tag
text>content],[<tag text>content],...,[<tag text>]

        //  handle all html table instructions

            if (tagBlock.equals(_p_rs_db.PF))                   //  an html "<" (PF)
tag prefix?
            {
                String tagText =
                    tagBlocks.nextToken(_p_rs_db.SF);           //  should be tag text

                if (tagBlocks.hasMoreTokens())
                    tagBlock = tagBlocks.nextToken();           //  should be ">" (SF)
tag suffix
                else
                    tagBlock = _p_rs_db.MT;                     //  syntax error

                if (tagBlock.equals(_p_rs_db.SF))               //  the normal case
                {
                    addElement(tagText);
                    tagBlock = _p_rs_db.MT;                     //  tagText handled,
transfer text capture
                }
                else                                            //  recover from
syntax error
                {
                    tagBlock = _p_rs_db.PF + tagText + tagBlock;//  restore tagBlock
to body text
                }
            }

        //  handle all html text content

            if (!tagBlock.equals(_p_rs_db.PF)
            &&  !tagBlock.equals(_p_rs_db.MT))                    //  not an html tag?
must be content
            {
                addContent(tagBlock);
            }
        }
        tableSpecParsed = true;
    }

    private int prefColLoc(int c)
    {
        int x = insets.left;
        for (int i = 0; i < c; i++)
            x += prefColWidth(i);

        return x;
    }

    private int prefColLoc(_p_tc_db c)
    {
        return prefColLoc(c.colNum);
    }

    private int prefColWidth(int c)
    {
        int prefColWidth = 0;
        for (int r = 0; r < nRows; r++)
            if (tableCells[r][c] != null)
                prefColWidth = Math.max
                (   prefColWidth
                ,   tableCells[r][c].preferredSize().width
                /   tableCells[r][c].colSpan
                );

        return prefColWidth;
    }

    private int prefColWidth(_p_tc_db c)
    {
        return prefColWidth(c.colNum);
    }

    public Dimension preferredSize()
    {
        if (!layoutComplete)
            doLayout();

    //
    dbg("_p_tb_db("+theTableNumber+").preferredSize("+prefTableSize().width+","+prefTa
bleSize().height+")");

        return prefTableSize();
    }

    public int prefRowHeight(int r)
    {
        int prefRowHeight = 0;
        for (int c = 0; c < nCols; c++)
            if (tableCells[r][c] != null)
                prefRowHeight = Math.max
                (   prefRowHeight
                ,   tableCells[r][c].preferredSize().height
                /   tableCells[r][c].rowSpan
                );

        return prefRowHeight;
    }

   private int prefRowHeight(_p_tc_db c)
    {
        return prefRowHeight(c.rowNum);
    }

    private int prefRowLoc(int r)
    {
        int y = insets.top;
        for (int j = 0; j < r; j++)
            y += prefRowHeight(j);

        return y;
    }

    private int prefRowLoc(_p_tc_db c)
    {
        return prefRowLoc(c.rowNum);
    }

    public Dimension prefTableSize()
    {
        Dimension d = new Dimension();

        for (int c = 0; c < nCols; c++)
            d.width += prefColWidth(c);

        for (int r = 0; r < nRows; r++)
            d.height += prefRowHeight(r);

        d.width += insets.left + insets.right;
        d.height += insets.top + insets.bottom;

        return d;
    }

    public synchronized void reshape(int x, int y, int width, int height)
    {
    //  dbg("_p_tb_db("+theTableNumber+").reshape("+x+", "+y+", "+width+",
"+height+")");

        super.reshape(x, y, width, height);

        if (!layoutComplete)
            doLayout();

        int iW = insets.left + insets.right;
        int iH = insets.top + insets.bottom;

        if (width > iW  && height > iH
        && (width != tableSize.width || height != tableSize.height))
            this.reshapeAll(width, height);

        repaint();
    }

    public final void reshapeAll()
    {
    //  dbg("_p_tb_db("+theTableNumber+").reshapeAll()");

        _p_tc_db firstCell = null;
        for (int j = 0; j < nRows-1 && firstCell == null; j++)
            for (int i = 0; i < nCols && firstCell == null; i++)
                firstCell = tableCells[j][i];

        _p_rs_db firstPanel = null;
        if (firstCell != null && firstCell.countComponents() > 0)
            if (firstCell.getComponent(0) instanceof _p_rs_db)
```

```
                firstPanel = (_p_rs_db)firstCell.getComponent(0);

        int sW =
            this.size().width == 0
            ?   Math.min
                (   getToolkit().getScreenSize().width
                ,   (   firstPanel != null
                    ?   firstPanel.standardLineWidth()
                    :   180
                    ) * nCols
                )
            :   this.size().width
            ;

        int sH =
            this.size().height == 0
            ?   getToolkit().getScreenSize().height
            :   this.size().height
            ;

        reshapeAll(sW, sH);
    }

    public final void reshapeAll(int sW, int sH)
    {
    //  dbg("_p_tb_db("+theTableNumber+").reshapeAll("+sW+", "+sH+")");

        int[] x = new int[nCols];   x[0] = insets.left;
        int[] y = new int[nRows];   y[0] = insets.top;
        int[] w = new int[nCols];
        int[] h = new int[nRows];

        boolean zoomableCells = true;           //  optimistic, to be sure

        Dimension p = prefTableSize();
        Dimension m = minTableSize();

        if (sW < m.width)                       //  at least 1 col set below min width
        {
            zoomableCells = false;              //  set false to zoom table instead

            for (int i = 0; i < nCols-1; i++)
            {
                x[i+1] = Math.min
                (   sW - insets.right           //  just truncate final column(s)
                ,   x[i] + minColWidth(i)
                );
                w[i] = x[i+1] - x[i];           //  result is always non-negative
            }

            w[nCols-1] = Math.max               //  the last column
            (   0
            ,   Math.min
                (   minColWidth(nCols-1)
                ,   sW - insets.right - x[nCols-1]
                )
            );
        }
        else
        if (sW < p.width)                       //  at least 1 col set below pref
width
        {
            zoomableCells = false;              //  set false to zoom table instead

            int[] z = new int[nCols];
            int nW = sW - insets.left - insets.right;

            for (int i = 0; i < nCols; i++)
            {
                z[i] = minColWidth(i);
                w[i] = prefColWidth(i);
                nW -= w[i];
            }
            while (nW < 0)                          //  decrement last column with most
slack
            {
                int xs = 0, xI = 0;
                for (int i = 0; i < nCols; i++)
                    if (xs < w[i] - z[i])           //  column slack = excess over minimum
width
                        xs = w[i] - z[xI = i];      //  get the column number
                w[xI]--;                            //  narrow the target column
                nW++;                               //  overshoot value increments closer
to zero
            }
            for (int i = 0; i < nCols-1; i++)
                x[i+1] = x[i] + w[i];
        }
        else                                        //  pref width is narrower than spec'd
width
        {
            for (int i = 0; i < nCols-1; i++)
                x[i+1] = x[i] + (w[i] = prefColWidth(i));

            w[nCols-1] = prefColWidth(nCols-1);
        }

        if (sH < m.height)                          //  at least 1 row set below min
height
        {
            zoomableCells = false;                  //  set false to zoom table instead

            for (int j = 0; j < nRows-1; j++)
            {
                y[j+1] = Math.min
                (   sH - insets.bottom              //  just truncate final row(s)
                ,   y[j] + minRowHeight(j)
                );
                h[j] = y[j+1] - y[j];               //  result is always non-negative
            }

            h[nRows-1] = Math.max                   //  the last row
            (   0
            ,   Math.min
                (   minRowHeight(nRows-1)
                ,   sH - insets.bottom - y[nRows-1]
                )
            );
        }
        else
        if (sH < p.height)                          //  at least 1 row set below pref
height
        {
            zoomableCells = false;                  //  set false to zoom table instead

            int[] z = new int[nRows];
            int nH = sH - insets.top - insets.bottom;

            for (int j = 0; j < nRows; j++)
            {
                z[j] = minRowHeight(j);
                h[j] = prefRowHeight(j);
                nH -= h[j];
            }
            while (nH < 0)                          //  decrement last row with most slack
            {
                int xs = 0, xJ = 0;
                for (int j = 0; j < nRows; j++)
                    if (xs < h[j] - z[j])           //  row slack = excess over minimum
height
```

```
                        xs = h[j] - z[xJ = j];  //  get the row number
                h[xJ]--;                        //  narrow the target row
                nH++;                           //  overshoot value increments closer
to zero
            }
            for (int j = 0; j < nRows-1; j++)
                y[j+1] = y[j] + h[j];
        }
        else                                    //  pref height is shorter than spec'd
height
        {
            for (int j = 0; j < nRows-1; j++)
                y[j+1] = y[j] + (h[j] = prefRowHeight(j));

            h[nRows-1] = prefRowHeight(nRows-1);
        }

        this.setZoomable(zoomableCells);

        for (int j = 0; j < nRows; j++)
        {
            for (int i = 0; i < nCols; i++) if (tableCells[j][i] != null)
            {
                _p_tc_db c = tableCells[j][i];

                int cWide = 0, cN = c.colNum;

                for (int n = 0; n < c.colSpan && n < nCols - cN; n++)
                    cWide += w[cN + n];

                int cHigh = 0, rN = c.rowNum;
                for (int n = 0; n < c.rowSpan && n < nRows - rN; n++)
                    cHigh += h[rN + n];

            //  dbg("_p_tb_db("+theTableNumber+").reshape cell["+j+"]["+i+"]:
("+x[cN]+", "+y[rN]+", "+cWide+", "+cHigh+")");

                c.reshape
                (   x[cN]
                ,   y[rN]
                ,   cWide
                ,   cHigh
                );
            }
        }
        tableSize = new Dimension(sW, sH);

    //  dbg("_p_tb_db("+theTableNumber+").reshapeAll("+sW+", "+sH+") complete: table
prefSize = "+p.width+"x"+p.height+"; miniSize = "+m.width+"x"+m.height);
    }

    public final void setFontSizeIndex(int newSizeIndex)
    {
    //  dbg("_p_tb_db("+theTableNumber+").setFontSizeIndex("+newSizeIndex+"): # comps
= "+countComponents());

        this.fontSizeIndex = newSizeIndex;
        for (int i = 0; i < this.countComponents(); i++)
        {
            if (this.getComponent(i) instanceof _p_tc_db)
            {
                ((_p_tc_db)this.getComponent(i)).
                    setFontSizeIndex(fontSizeIndex);
            }
        }
    }

    public void setZoomable(boolean zoomableCells)
    {
```

```
    //  dbg("_p_tb_db("+theTableNumber+").setZoomable("+zoomableCells+"): # comps =
"+countComponents());

        for (int i = 0; i < this.countComponents(); i++)
        {
            if (this.getComponent(i) instanceof _p_tc_db)
            {
                ((_p_tc_db)this.getComponent(i)).
                    setZoomable(zoomableCells);
            }
        }
    }

    public synchronized void update(Graphics g)
    {
    //  fillRect is NOT performed here to eliminate blank screen boredom during
offscreen drawing

    //  g.setColor(getBackground());
    //  g.fillRect(0, 0, width, height);
    //  g.setColor(getForeground());

        paint(g);
    }

    public void validate()
    {
    //  dbg("_p_tb_db("+theTableNumber+").validate()");

        if (!this.isValid() && !layoutComplete)
            doLayout();

        super.validate();
    }

    public _p_tb_db(String tableSpec, Integer theTableNumber, ez_html ez_HTML)
    {
    //  dbg("new _p_tb_db("+tableSpec+", "+theTableNumber+")");

        this.ez_HTML = ez_HTML;
        this.tableSpec = tableSpec;
        this.theTableNumber = theTableNumber;
        this.setLayout(null);

        insets.top      = vSpace + 1 + border + cellSpacing/2;
        insets.left     = hSpace + 1 + border + cellSpacing/2;
        insets.bottom   = vSpace + 1 + border + cellSpacing - cellSpacing/2;
        insets.right    = hSpace + 1 + border + cellSpacing - cellSpacing/2;
    }
}

/**
 *  %W% %E% Everett Stoub
 *
 *  Copyright (c) '96-'97 ION Web Products, Inc. All Rights Reserved.
 *
 *  an object to store table cell data
 *
 *  this class is not downloaded unless needed
 *
 *  @author Everett Stoub %I%, %G%
 */
import java.awt.*;
import java.util.*;

public class _p_tc_db extends Panel
{
    public Color    cellBg      = null;
    public Color    cellFg      = null;
    public int      align       = _p_rs_db.LEFT;
    public int      vAlign      = _p_rs_db.CENTER;
    public int      minWidth    = 0;
    public int      minHeight   = 0;
    public int      colSpan     = 1;
    public int      rowSpan     = 1;
    public int      colNum      = 1;
    public int      rowNum      = 1;
    public int      border      = 0;
    public int      cellPadding = 0;
    public int      cellSpacing = 0;
    public boolean  isData      = true;
    public String   content     = _p_rs_db.MT;

    private final void dbg(String s)
    {
        if (debug) System.out.println(s);
    }
    private boolean debug = false;          //  Print debugging info?

    public boolean mouseDown(Event evt, int x, int y)
    {
    //  dbg("_p_tc_db.mouseDown("+x+","+y+")");

        return false;
    }

    public final void setFontSizeIndex(int newSizeIndex)
    {
    //  dbg("_p_tc_db().setFontSizeIndex("+newSizeIndex+"): # comps =
"+countComponents());

        for (int i = 0; i < this.countComponents(); i++)
        {
            if (this.getComponent(i) instanceof _p_rs_db)
                ((_p_rs_db)this.getComponent(i)).
                    setFontSizeIndex(newSizeIndex);
            else
            if (this.getComponent(i) instanceof _p_tb_db)
                ((_p_tb_db)this.getComponent(i)).
                    setFontSizeIndex(newSizeIndex);
        }
    }

    public void setZoomable(boolean zoomable)
    {
    //  dbg("_p_tc_db().setZoomable("+zoomable+"): # comps = "+countComponents());

        for (int i = 0; i < this.countComponents(); i++)
        {
            if (this.getComponent(i) instanceof _p_rs_db)
            {
                ((_p_rs_db)this.getComponent(i)).
                    setZoomable(zoomable);
            }
            else
            if (this.getComponent(i) instanceof _p_tb_db)
            {
                ((_p_tb_db)this.getComponent(i)).
                    setZoomable(zoomable);
            }
        }
    }

    public synchronized void update(Graphics g)
    {
    //  fillRect is NOT performed here to eliminate blank screen boredom during
offscreen drawing

    //  g.setColor(getBackground());
    //  g.fillRect(0, 0, width, height);
    //  g.setColor(getForeground());

        paint(g);
    }

    public _p_tc_db
    (   Color    cellBg
    ,   Color    cellFg
    ,   int      align
    ,   int      vAlign
    ,   int      minWidth
    ,   int      colSpan
    ,   int      rowSpan
    ,   int      minHeight
    ,   int      border
    ,   int      cellPadding
    ,   int      cellSpacing
    ,   boolean isData  //  false if header, true if data
    )
    {
        this.cellBg          = cellBg;
        this.cellFg          = cellFg;
        this.align           = align;
        this.vAlign          = vAlign;
        this.minWidth        = minWidth;
        this.colSpan         = colSpan;
        this.rowSpan         = rowSpan;
        this.minHeight       = minHeight;
        this.border          = border;
        this.cellPadding     = cellPadding;
        this.cellSpacing     = cellSpacing;
        this.isData          = isData;

        this.setLayout(new BorderLayout());
    }
}

/**
 *  %W% %E% Everett Stoub
 *
 *  Copyright (c) '96-'97 ION Web Products, Inc. All Rights Reserved.
 *
 *  an object to handle html horizontal rules
 *
 *  this class is not downloaded unless needed
 *
 *  @author Everett Stoub %I%, %G%
 */
import java.awt.*;
import java.io.*;
import java.net.*;

public class _o_hr_db extends Object
{
    public static int placeRule
    (   String tagText
    ,   _p_rs_db panel
    ,   int addIndexValue
    ,   Graphics g
    )
    {
        int addIndex = addIndexValue;
        int itsSize = _o_tg_db.getTagAttribInt(tagText, panel.SZ, 3);
        int itsWidth = _o_tg_db.getTagWidth(tagText,panel.effectiveWidth());

        addIndex += panel.paintRow(g,0);                          //  try to punch out
pending line elements
        if (!panel.startNewParagraph(1))
        {
                return -addIndex;                                    //  no more room on
the page
            }
            while (panel.yb + itsSize > panel.ye)
            {
                if (panel.lastColumn)
                {
                    return -addIndex;                                //  no more columns
left to fill
                }
                panel.nextColumn();
            }
            if (itsSize > 0 && itsWidth > 0)
            {
                int itsAlign = _o_tg_db.getTagAlignment(tagText, panel.CENTER);
                int itsStart = panel.xb + panel.indent(false) +
itsAlign*(panel.effectiveWidth() - itsWidth)/2;

                float[] hsbvals = new float[3];
                hsbvals = Color.RGBtoHSB(panel.getBackground().getRed(),
panel.getBackground().getGreen(), panel.getBackground().getBlue(), hsbvals);
                if (g != null)
                {
                    if (hsbvals[2] > 0.5f)
                    {
                        g.setColor(panel.getBackground().darker().darker());
                    }
                    else
                    {
                        g.setColor(panel.getBackground().brighter().brighter());
                    }
                    if (tagText.indexOf("noshade") > -1)
                    {

    g.fillRoundRect(itsStart,panel.yb,itsWidth,itsSize,itsSize,itsSize);
                    }
                    else
                    {
                        g.draw3DRect(itsStart,panel.yb,itsWidth,itsSize,false);
                    }
                }
                panel.yb += itsSize;
                if (!panel.startNewParagraph(1))
                {
                    return -addIndex;
                }
            }
    //  textAlign = LEFT;                                //  it's supposed to be reset
this way, but PageMill does not!
            return addIndex;
        }
}

/**
 *  %W% %E% Everett Stoub
 *
 *  Copyright (c) '96-'97 ION Web Products, Inc. All Rights Reserved.
 *
 *  an object to handle html lists
 *
 *  this class is not downloaded unless needed
 *
 *  @author Everett Stoub %I%, %G%
 */
import java.awt.*;
import java.io.*;
import java.net.*;

public class _o_li_db extends Object
{
    public final static String  ST          = "start";

//  unordered list bullet types

    public final static int     DISC        =   1;
    public final static int     CIRCLE      =   2;
    public final static int     BLOCK       =   3;
    public final static int     SQUARE      =   4;

//  ordered list numbering types

    public final static int     UCROMAN     =   1;      //  list type "I"
    public final static int     UCLETTER    =   2;      //  list type "A"
    public final static int     ARABIC      =   3;      //  list type "1"
    public final static int     LCROMAN     =   4;      //  list type "i"
    public final static int     LCLETTER    =   5;      //  list type "a"

    public static int addItem
    (   String tagText
    ,   String itsText
    ,   _p_rs_db panel
    ,   int addIndexValue
    ,   Graphics g
    )
    {
    //  dbg("_o_li_db.addItem(["+itsText+"],panel,"+addIndexValue+",g)");

        int addIndex = addIndexValue;

        if (panel.listAccount.size() > 0)
        {
            addIndex += panel.paintRow(g,0);            //  try to punch out pending
line elements
            if (!panel.startNewParagraph(0))
            {
                return -addIndex;
            }
            if (tagText.startsWith(panel.LI))
            {
                Point itsItem = (Point)panel.listAccount.firstElement();
                itsItem.x = _o_tg_db.getTagAttribInt
                (   tagText                             //  optional start value
                ,   ST
                ,   itsItem.x                           //  increment its counter in
panel.paintRow
                );
                itsItem.y = _o_li_db.getTagAttribTypeInt
                (   itsText                             //  as asserted by user
                ,   itsItem.y                           //  inherited list element
type
                );
            }
            else                                        //  a definition list element?
            {
                panel.termDefinition
                    = tagText.equals(panel.DD);         //  if true: added indentation
            }
            panel.newListItem = true;
        }
        return addIndex;
    }

    public static int addList
    (   String tagText
    ,   String itsText
    ,   _p_rs_db panel
    ,   int addIndexValue
    ,   Graphics g
    )
    {
        int addIndex = addIndexValue;

        addIndex += panel.paintRow(g,0);                // try to punch out pending
line elements
        if (!panel.startNewParagraph((panel.listAccount.size() > 0)?0:1))
        {
            return -addIndex;
        }
        if (tagText.startsWith(panel.OL))
        {
            panel.orderedList = true;
        }
        else
        {
            panel.unorderList = true;
        }
        if (tagText.equals(panel.DL))
        {
            panel.listAccount.insertElementAt(new Point(1,0),0);
        }
        else
        {
            int litIndex = _o_tg_db.getTagAttribInt(tagText, ST, 1) - 1;
            int listType = _o_li_db.getTagAttribTypeInt
            (   itsText
            ,   panel.listAccount.size() % (panel.unorderList?4:5) + 1
            );

            Point listElement =
                new Point
                (   litIndex                              // start value
                ,   listType                              // element type
                );
            panel.listAccount.insertElementAt(listElement,0);
        }

        return addIndex;
    }

    public static int endList
    (   String tagText
    ,   String itsText
    ,   _p_rs_db panel
    ,   int addIndexValue
    ,   Graphics g
    )
    {
        int addIndex = addIndexValue;

        addIndex += panel.paintRow(g,0);                    // try to punch out
pending line elements
        if (!panel.startNewParagraph((panel.listAccount.size() > 0)?0:1))
        {
            return -addIndex;
        }
        int itsSize = panel.listAccount.size();
        if (itsSize == 1)
        {
            if (tagText.equals(panel.SL+panel.OL))
            {
                panel.orderedList = false;
            }
            else
            {
                panel.unorderList = false;
            }
        }
        if (itsSize > 0)
        {
            panel.listAccount.removeElementAt(0);           // LIFO list
        }

        return addIndex;
    }

    private static final void dbg(String s)
    {
        if (debug) System.out.println(s);
    }
    private static boolean debug = false;    //  Print debugging info?

    public static void drawBullet
    (   _p_rs_db panel
    ,   int x
    ,   int y
    ,   Graphics g
    )
    {
        if (panel.listAccount.size() == 0
        || !(panel.listAccount.firstElement() instanceof Point)) return;

        Point itsItem = (Point)panel.listAccount.firstElement();
        int D = (panel.textHeight+3)/4;
        if (g != null && itsItem.x > 0) switch (itsItem.y)
        {
            case DISC:      g.fillOval(x - 2*D - 3, y - D - 2, D + 1, D + 1);
            case CIRCLE:    g.drawOval(x - 2*D - 3, y - D - 2, D + 1, D + 1);   break;
            case BLOCK:     g.fillRect(x - 2*D - 1, y - D - 2, D, D);
            case SQUARE:    g.drawRect(x - 2*D - 1, y - D - 2, D, D);            break;
        }
        panel.newListItem = false;
    }

    public static void drawLabel
    (   _p_rs_db panel
    ,   int x
    ,   int y
    ,   Graphics g
    )
    {
        if (panel.listAccount.size() == 0
        || !(panel.listAccount.firstElement() instanceof Point)) return;

        Point itsItem = (Point)panel.listAccount.firstElement();
        String itsLabel = ". ";
        if (itsItem.x > 0) switch (itsItem.y)
        {
            case UCROMAN:   //  up to 100 entries before starting over
            case LCROMAN:
                {
                    String lab = panel.MT;
                    switch ((itsItem.x - 1) % 10 + 1)
                    {
                        case  1: lab = "i";             break;
                        case  2: lab = "ii";            break;
                        case  3: lab = "iii";           break;
                        case  4: lab = "iv";            break;
                        case  5: lab = "v";             break;
                        case  6: lab = "vi";            break;
                        case  7: lab = "vii";           break;
                        case  8: lab = "viii";          break;
                        case  9: lab = "ix";            break;
                    }
                    switch ((itsItem.x/10 - 1) % 10 + 1)
                    {
                        case  1: lab = "x"      + lab;  break;
                        case  2: lab = "xx"     + lab;  break;
                        case  3: lab = "xxx"    + lab;  break;
                        case  4: lab = "xl"     + lab;  break;
                        case  5: lab = "l"      + lab;  break;
                    case 6: lab = "lx"     + lab;  break;
                    case 7: lab = "lxx"    + lab;  break;
                    case 8: lab = "lxxx"   + lab;  break;
                    case 9: lab = "xc"     + lab;  break;
                }
                if (itsItem.y == UCROMAN)
                    lab = lab.toUpperCase();
                itsLabel = lab + itsLabel;
            }
            break;
        case UCLETTER:  //  up to 26**2 entries before starting over
        case LCLETTER:
            {
                char[] lab = {(char)(97 + (itsItem.x - 1) % 26)};
                itsLabel = (new String(lab)) + itsLabel;
            }
            if (itsItem.x > 26)
            {
                char[] lab = {(char)(97 + (itsItem.x/26 - 1) % 26)};
                itsLabel = (new String(lab)) + itsLabel;
            }
            if (itsItem.y == UCLETTER)
                itsLabel = itsLabel.toUpperCase();
            break;
        case ARABIC:
            itsLabel = itsItem.x + itsLabel;
            break;
    }

    if (g != null) g.drawString(itsLabel, x -
panel.textMetrics.stringWidth(itsLabel), y);
    panel.newListItem = false;
  }

  public static final int getTagAttribTypeInt
  (   String itsText
  ,   int itsDefault
  )
  {
      String itsLabel = _o_tg_db.getTagAttribString(itsText, _p_rs_db.TP);

      if (itsLabel == null)
      {
          return itsDefault;
      }

      String lowLabel = itsLabel.toLowerCase();

      if (lowLabel.equals("circle"))  return CIRCLE;      else
      if (lowLabel.equals("square"))  return SQUARE;      else
      if (lowLabel.equals("disc"))    return DISC;        else

      if (itsLabel.equals("1"))       return ARABIC;      else
      if (itsLabel.equals("A"))       return UCLETTER;    else
      if (itsLabel.equals("a"))       return LCLETTER;    else
      if (itsLabel.equals("I"))       return UCROMAN;     else
      if (itsLabel.equals("i"))       return LCROMAN;     else

      return itsDefault;
  }
}

/**
 *  %W% %E% Everett Stoub
 *
 *  Copyright (c) '96-'97 ION Web Products, Inc. All Rights Reserved.
 *
 *  platform-specific html character entity converter
 *
 *  Character Entity parsing honors the following characters
 *
 *    NOTE: each JVM will yield some variations due to internal font implementations
 *
 *                                                       CHARACTER REMAPPING
 *    NAMED        NUMERIC   DESCRIPTION                  ALL     MAC     ALT
 *    =====        =======   ========================     ===     ===     ===
 *    &quot;       &#034;    double quotation mark
 *    &amp;        &#038;    ampersand
 *    &lt;         &#060;    less than sign
 *    &gt;         &#062;    greater than sign
 *                 &#130;    comma                        44
 *                 &#131;    florin                               159     102
 *                 &#132;    right double quote                   155      34
 *                 &#133;    ellipsis                             138     "..."
 *                 &#134;    dagger                               221     n/a
 *                 &#135;    double dagger                        253     n/a
 *                 &#136;    circumflex                           150     n/a
 *                 &#137;    permil                               148     "0/00"
 *                 &#138;    underscore                   95
 *                 &#139;    less than sign               60
 *                 &#140;    capital OE ligature                  145     "OE"
 *                 &#145;    left single quote                    140      39
 *                 &#146;    right single quote                   140      39
 *                 &#147;    left double quote                    155      34
 *                 &#148;    right double quote                   155      34
 *                 &#149;    bullet                               128     183
 *                 &#150;    em dash                              173      45
 *                 &#151;    en dash                              139      45
 *                 &#152;    tilde                        126
 *                 &#153;    trademark                            129     "(TM)"
 *                 &#154;    underscore                   95
 *                 &#155;    greater than sign            62
 *                 &#156;    small OE ligature            "oe"
 *                 &#159;    capital Y umlaut                     141     n/a
 *    &nbsp;       &#160;    nonbreaking space
 *    &iexcl;      &#161;    inverted exclamation point
 *    &cent;       &#162;    cents sign
 *    &pound;      &#163;    pound sign
 *    &curren;     &#164;    general currency sign
 *    &yen;        &#165;    Yen sign
 *    &brvbar;     &#166;    broken vertical bar                  "|"
 *    &sect;       &#167;    section mark
 *    &uml;        &#168;    umlaut
 *    &copy;       &#169;    copyright mark
 *    &ordf;       &#170;    feminine ordinal
 *    &laguo;      &#171;    left angle quote
 *    &not;        &#172;    not sign
 *    &shy;        &#173;    soft hyphen
 *    &reg;        &#174;    registered trade mark
 *    &macr;       &#175;    macron accent
 *    &deg;        &#176;    degree sign
 *    &plusmn;     &#177;    plus or minus
 *    &sup2;       &#178;    superscript 2                        "2"
 *    &sup3;       &#179;    superscript 3                        "3"
 *    &acute;      &#180;    acute accent
 *    &micro;      &#181;    micro sign (Greek mu)
 *    &para;       &#182;    paragraph sign
 *    &middot;     &#183;    middle dot mark
 *    &cedil;      &#184;    cedilla
 *    &sup1;       &#185;    superscript 1                        "1"
 *    &ordm;       &#186;    masculine ordinal
 *    &raquo;      &#187;    right angle quote
 *    &frac14;     &#188;    fraction one-fourth                  "1/4"
 *    &frac12;     &#189;    fraction one-half                    "1/2"
 *    &frac34;     &#190;    fraction three-fourths               "3/4"
 *    &iquest;     &#191;    inverted question mark
 *    &Agrave;     &#192;    cap A with grave accent
 *    &Aacute;     &#193;    cap A with acute accent
 *    &Acirc;      &#194;    cap A with circumflex accent
 *    &Atilde;     &#195;    cap A with tilde accent
 *        &Auml;       &#196;  cap A with unlaut accent
 *        &Aring;      &#197;  cap A with ring accent
 *        &AElig;      &#198;  cap AE ligature
 *        &Ccedil;     &#199;  cap cedilla
 *        &Egrave;     &#200;  cap E with grave accent
 *        &Eacute;     &#201;  cap E with acute accent
 *        &Ecirc;      &#202;  cap E with circumflex accent
 *        &Euml;       &#203;  cap E with unlaut accent
 *        &Igrave;     &#204;  cap I with grave accent
 *        &Iacute;     &#205;  cap I with acute accent
 *        &Icirc;      &#206;  cap I with circumflex accent
 *        &Iuml;       &#207;  cap I with unlaut accent
 *        &ETH;        &#208;  cap eth, Icelandic                 127
 *        &Ntilde;     &#209;  cap N with tilde accent
 *        &Ograve;     &#210;  cap O with grave accent
 *        &Oacute;     &#211;  cap O with acute accent
 *        &Ocirc;      &#212;  cap O with circumflex accent
 *        &Otilde;     &#213;  cap O with tilde accent
 *        &Ouml;       &#214;  cap O with unlaut accent
 *        &times;      &#215;  multiply sign                      183
 *        &Oslash;     &#216;  cap O with slash
 *        &Ugrave;     &#217;  cap U with grave accent
 *        &Uacute;     &#218;  cap U with acute accent
 *        &Ucirc;      &#219;  cap U with circumflex accent
 *        &Uuml;       &#220;  cap U with unlaut accent
 *        &Yacute;     &#221;  cap U with acute accent             89
 *        &THORN;      &#222;  cap THORN, Icelandic               127
 *        &szlig;      &#223;  small sz ligature, German
 *        &agrave;     &#224;  small A with grave accent
 *        &aacute;     &#225;  small A with acute accent
 *        &acirc;      &#226;  small A with circumflex accent
 *        &atilde;     &#227;  small A with tilde accent
 *        &auml;       &#228;  small A with unlaut accent
 *        &aring;      &#229;  small A with ring accent
 *        &aelig;      &#230;  small AE ligature
 *        &ccedil;     &#231;  small cedilla
 *        &egrave;     &#232;  small E with grave accent
 *        &eacute;     &#233;  small E with acute accent
 *        &ecirc;      &#234;  small E with circumflex accent
 *        &euml;       &#235;  small E with unlaut accent
 *        &igrave;     &#236;  small I with grave accent
 *        &iacute;     &#237;  small I with acute accent
 *        &acirc;      &#238;  small I with circumflex accent
 *        &iuml;       &#239;  small I with unlaut accent
 *        &eth;        &#240;  small eth, Icelandic               143
 *        &ntilde;     &#241;  small N with tilde accent
 *        &ograve;     &#242;  small O with grave accent
 *        &oacute;     &#243;  small O with acute accent
 *        &ocirc;      &#244;  small O with circumflex accent
 *        &otilde;     &#245;  small O with tilde accent
 *        &ouml;       &#246;  small O with unlaut accent
 *        &divide;     &#247;  divide sign
 *        &oslash;     &#248;  small O with slash
 *        &ugrave;     &#249;  small U with grave accent
 *        &uacute;     &#250;  small U with acute accent
 *        &ucirc;      &#251;  small U with circumflex accent
 *        &uuml;       &#252;  small U with unlaut accent
 *        &yacute;     &#253;  small Y with acute accent          121
 *        &thorn;      &#254;  small THORN, Icelandic             127
 *        &yuml;       &#255;  small Y with unlaut accent
 *
 *
 *  @author Everett Stoub %I%, %G%
 */
import java.awt.*;
import java.util.*;

public class _o_nt_db extends Object
{
    public String convertCharacterEntities(String itsText)
    {
    //  dbg("_o_nt_db.convertCharacterEntities(["+itsText+"])");

        if (itsText == null)
            return itsText;

        boolean mac = System.getProperty("os.name").equals("macos");

        StringTokenizer entities = new StringTokenizer(itsText,"&",true);
        StringBuffer newText = new StringBuffer(70);

        while (entities.hasMoreTokens())
        {
            String entity =                          //  [body text],[&][#nnn;body
text],...,[&][#nnn;body text]
                entities.nextToken("&");

            if (entity.equals("&")
            &&  entities.hasMoreTokens())
            {
                String eText =                       //  should be character entity
text #nnn or name
                    entities.nextToken(";");

                if (entities.hasMoreTokens())
                    entity = entities.nextToken();   //  should be ";" if eText is
valid character entity value
                else
                    entity = _p_rs_db.MT;            //  avoid making things up

                if (entity.equals(";"))              //  replace character entity with
local character (set)
                {
                    int e = 0;
                    if (!eText.startsWith("#"))      //  named entity: assign numeric
entity value
                    {
                        if (eText.equals("quot"))    e =  34;    else
                        if (eText.equals("amp"))     e =  38;    else
                        if (eText.equals("lt"))      e =  60;    else
                        if (eText.equals("gt"))      e =  62;    else
                        if (eText.equals("nbsp"))    e = 160;    else
                        if (eText.equals("iexcl"))   e = 161;    else
                        if (eText.equals("cent"))    e = 162;    else
                        if (eText.equals("pound"))   e = 163;    else
                        if (eText.equals("curren"))  e = 164;    else
                        if (eText.equals("yen"))     e = 165;    else
                        if (eText.equals("brvbar"))  e = 166;    else
                        if (eText.equals("sect"))    e = 167;    else
                        if (eText.equals("uml"))     e = 168;    else
                        if (eText.equals("copy"))    e = 169;    else
                        if (eText.equals("ordf"))    e = 170;    else
                        if (eText.equals("laquo"))   e = 171;    else
                        if (eText.equals("not"))     e = 172;    else
                        if (eText.equals("shy"))     e = 173;    else
                        if (eText.equals("reg"))     e = 174;    else
                        if (eText.equals("macr"))    e = 175;    else
                        if (eText.equals("deg"))     e = 176;    else
                        if (eText.equals("plusmn"))  e = 177;    else
                        if (eText.equals("sup2"))    e = 178;    else
                        if (eText.equals("sup3"))    e = 179;    else
                        if (eText.equals("acute"))   e = 180;    else
                        if (eText.equals("micro"))   e = 181;    else
                        if (eText.equals("para"))    e = 182;    else
                        if (eText.equals("middot"))  e = 183;    else
                        if (eText.equals("cedil"))   e = 184;    else
                        if (eText.equals("sup1"))    e = 185;    else
                        if (eText.equals("ordm"))    e = 186;    else
                        if (eText.equals("raquo"))   e = 187;    else
                        if (eText.equals("frac14"))  e = 188;    else
if (eText.equals("frac12"))  e = 189;    else
if (eText.equals("frac34"))  e = 190;    else
if (eText.equals("iquest"))  e = 191;    else
if (eText.equals("Agrave"))  e = 192;    else
if (eText.equals("Aacute"))  e = 193;    else
if (eText.equals("Acirc"))   e = 194;    else
if (eText.equals("Atilde"))  e = 195;    else
if (eText.equals("Auml"))    e = 196;    else
if (eText.equals("Aring"))   e = 197;    else
if (eText.equals("AElig"))   e = 198;    else
if (eText.equals("Ccedil"))  e = 199;    else
if (eText.equals("Egrave"))  e = 200;    else
if (eText.equals("Eacute"))  e = 201;    else
if (eText.equals("Ecirc"))   e = 202;    else
if (eText.equals("Euml"))    e = 203;    else
if (eText.equals("Igrave"))  e = 204;    else
if (eText.equals("Iacute"))  e = 205;    else
if (eText.equals("Icirc"))   e = 206;    else
if (eText.equals("Iuml"))    e = 207;    else
if (eText.equals("ETH"))     e = 208;    else
if (eText.equals("Ntilde"))  e = 209;    else
if (eText.equals("Ograve"))  e = 210;    else
if (eText.equals("Oacute"))  e = 211;    else
if (eText.equals("Ocirc"))   e = 212;    else
if (eText.equals("Otilde"))  e = 213;    else
if (eText.equals("Ouml"))    e = 214;    else
if (eText.equals("times"))   e = 215;    else
if (eText.equals("Oslash"))  e = 216;    else
if (eText.equals("Ugrave"))  e = 217;    else
if (eText.equals("Uacute"))  e = 218;    else
if (eText.equals("Ucirc"))   e = 219;    else
if (eText.equals("Uuml"))    e = 220;    else
if (eText.equals("Yacute"))  e = 221;    else
if (eText.equals("THORN"))   e = 222;    else
if (eText.equals("szlig"))   e = 223;    else
if (eText.equals("agrave"))  e = 224;    else
if (eText.equals("aacute"))  e = 225;    else
if (eText.equals("acirc"))   e = 226;    else
if (eText.equals("atilde"))  e = 227;    else
if (eText.equals("auml"))    e = 228;    else
if (eText.equals("aring"))   e = 229;    else
if (eText.equals("aelig"))   e = 230;    else
if (eText.equals("ccedil"))  e = 231;    else
if (eText.equals("egrave"))  e = 232;    else
if (eText.equals("eacute"))  e = 233;    else
if (eText.equals("ecirc"))   e = 234;    else
if (eText.equals("euml"))    e = 235;    else
if (eText.equals("igrave"))  e = 236;    else
if (eText.equals("iacute"))  e = 237;    else
if (eText.equals("acirc"))   e = 238;    else
if (eText.equals("iuml"))    e = 239;    else
if (eText.equals("eth"))     e = 240;    else
if (eText.equals("ntilde"))  e = 241;    else
if (eText.equals("ograve"))  e = 242;    else
if (eText.equals("oacute"))  e = 243;    else
if (eText.equals("ocirc"))   e = 244;    else
if (eText.equals("otilde"))  e = 245;    else
if (eText.equals("ouml"))    e = 246;    else
if (eText.equals("divide"))  e = 247;    else
if (eText.equals("oslash"))  e = 248;    else
if (eText.equals("ugrave"))  e = 249;    else
if (eText.equals("uacute"))  e = 250;    else
if (eText.equals("ucirc"))   e = 251;    else
if (eText.equals("uuml"))    e = 252;    else
if (eText.equals("yacute"))  e = 253;    else
if (eText.equals("yuml"))    e = 255;    else
{
    e =  32;    //  space
}
```

```
                    //  dbg("_o_nt_db.convertCharacterEntities(): the named character
entity &"+eText+"; was mapped to numeric code "+e);
                    }
                    else                            //  numeric entity: look after the
pound sign
                    {
                        try
                        {
                            e = Integer.parseInt(eText.substring(1));
                        }
                        catch (NumberFormatException ne)
                        {
                        //  dbg("_o_nt_db.convertCharacterEntities(): the numeric
entity &"+eText+"; was not parsed");
                            e = 160;                //  nonbreaking space
                        };
                    }

                //  character or string remapping (internal font mapping exceptions)

                    String g = null;
                    switch (e)
                    {
                        case 130: e =  44;      break;
                        case 138:
                        case 154: e =  95;      break;
                        case 139: e =  60;      break;
                        case 152: e = 126;      break;
                        case 155: e =  62;      break;
                        case 156: g = "oe";     break;
                    }
                    if (mac) switch (e)
                    {
                        case 131: e = 159;      break;
                        case 132:
                        case 147:
                        case 148: e = 155;      break;
                        case 133: e = 138;      break;
                        case 134: e = 221;      break;
                        case 135: e = 253;      break;
                        case 136: e = 150;      break;
                        case 137: e = 148;      break;
                        case 140: e = 145;      break;
                        case 145:
                        case 146: e = 140;      break;
                        case 149: e = 128;      break;
                        case 150: e = 173;      break;
                        case 151: e = 139;      break;
                        case 153: e = 129;      break;
                        case 159: e = 141;      break;
                        case 166: g = "|";      break;
                        case 178: g = "2";      break;
                        case 179: g = "3";      break;
                        case 185: g = "1";      break;
                        case 188: g = "1/4";    break;
                        case 189: g = "1/2";    break;
                        case 190: g = "3/4";    break;
                        case 208:
                        case 222:
                        case 254: e = 127;      break;
                        case 215: e = 183;      break;
                        case 221: e =  89;      break;
                        case 240: e = 143;      break;
                        case 253: e = 121;      break;
                    }
                    else switch (e)
                    {
                        case 131: e = 102;      break;
                        case 132:
                        case 147:
```

```
                        case 148: e =  34;      break;
                        case 133: g = "...";    break;
                        case 137: g = "0/00";   break;
                        case 140: g = "OE";     break;
                        case 145:
                        case 146: e =  39;      break;
                        case 149: e = 183;      break;
                        case 150:
                        case 151: e =  45;      break;
                        case 153: g = "(TM)";   break;
                    }
                    char[] sub = {(char)e};
                    newText.append((g == null? new String(sub):g));

                //  dbg("_o_nt_db.convertCharacterEntities(): the character entity
&"+eText+"; converted to ["+(g == null? new String(sub): g)+"]");
                }
                else                                    //  syntax error:
                {
                    newText.append("&").append(eText).append(entity);
                }
            }
            else
            {
                newText.append(entity);                 //  standard text
            }
        }
        return newText.toString();
    }

    public static final void dbg(String s)
    {
        if (debug) System.out.println(s);
    }
    private static boolean debug = false;   //  Print debugging info?
}

/**
 *  sample web page with applet tag
 */
<HTML>
<HEAD>
  <META NAME="GENERATOR" CONTENT="Adobe PageMill 2.0 Mac">
  <TITLE></TITLE>
</HEAD>
<BODY>

<APPLET CODE="ez_html.class" CODEBASE="ez_h_cls/" WIDTH="640" HEIGHT="480"
ALIGN="BOTTOM">
<PARAM NAME="fileName" VALUE="ezsample.htm">
<PARAM NAME="maxColumns" VALUE="4">
</APPLET>

</BODY>
</HTML>

/**
 *  ezsample.htm web page for applet
 */
<HTML>
<HEAD>
  <META NAME="GENERATOR" CONTENT="Adobe PageMill 2.0 Mac">
  <TITLE>EZ HTML Sample Article</TITLE>
</HEAD>
<BODY BGCOLOR="#e8e8e8">

<P>EZ HTML is an <FONT COLOR="#00BA00">applet</FONT> for delivering substantial
text and graphics content while enhancing your web page's appearance. A
mouse click on the applet will evoke a reader screen window, or if over
a hypertext link, will open the <A HREF="ez_html.html">selected page with
```

```
a really long hypertext label which can show up in multiple lines and across
column and page boundaries</A> in the browser.</P>

<P>NOTE: this is a test document, in which samples of various html features
are rendered by the EZ HTML applet for comparison with standard browser
rendering. The table cells below contains very little information (most
or all of each cell should be displayed). Any cell can be viewed in a reading
window by clicking in it. The status bar provides context-sensitive help
text. You can also click on a table or cell boundary to view the entire
table in a larger window. EZ HTML is still in alpha - not all features have
been implemented, and not all implemented features are fully debugged.</P>

<P><TABLE BORDER="1" CELLSPACING="0" CELLPADDING="5">
<CAPTION ALIGN="BOTTOM">Kumquat versus a poked eye, by gender</CAPTION>
<TR>
<TD COLSPAN="2" ROWSPAN="2"><FONT COLOR="#0000FF">Note: This table is taken from
<I>HTML,
The Definitive Guide,</I> by Chuck Musciano and Bill Kennedy, an O'Reilly
&amp; Associates, Inc. publication. The table is merely designed to illustrate
multiple-row and column headers, blank areas and other special aspects to
HTML table structures.</FONT></TD>
<TH COLSPAN="2"><I><FONT COLOR="#FF0000" SIZE=+2>Preference</FONT></I></TH></TR>
<TR>
<TH>Eating Kumquats</TH>
<TH>Poke In The Eye</TH></TR>
<TR ALIGN="CENTER">
<TH ROWSPAN="2">Gender</TH>
<TH>Male</TH>
<TD>73%</TD>
<TD>27%</TD></TR>
<TR ALIGN="CENTER">
<TH>Female</TH>
<TD>16%</TD>
<TD>84%</TD></TR>
</TABLE>
</P>

<P ALIGN=CENTER><FONT COLOR="#330099" SIZE=+1><HR NOSHADE>&quot;Click on
the applet&quot;<HR NOSHADE></FONT></P>

<P>The reader screen presentation of content is interactive and optimized
for readability. Font size can be changed by pressing up or down arrow keys,
or by clicking the Font+ or Font- buttons. Pages can be turned by pressing
home, left, or right keys, or by clicking the Home, Previous, or Next buttons.</P>

<P ALIGN=CENTER><FONT COLOR="#330099" SIZE=+1><HR NOSHADE>&quot;optimized
for readability&quot;<HR NOSHADE></FONT></P>

<P>One of the key advantages of this reader screen presentation is its careful
avoidance of a particular form of eye-strain known as optokinetic nystagmus.
Nystagmus, a reflexive jump in the muscles directing the eye, is triggered
by, for example, vertical scrolling of text on a computer screen.</P>

<P ALIGN=CENTER><IMG SRC="ez_eye.gif" WIDTH="74" HEIGHT="72" NATURALSIZEFLAG=
"3" ALIGN="BOTTOM"></P>

<P>As nystagmus strain increases, muscle jumps can become exaggerated and
even erratic, finally resulting in a perception of visual weariness. Nystagmus
is well-known as a side-effect of excessive alcohol: a simple road-side
test conducted by law officers can aid in determination of a possible DWI
offense. Other possible causes of nystagmus include general physical exhaustion
and illness.</P>

<P>While the term &quot;nystagmus&quot; may have been unfamiliar, the condition
is encountered in many circumstances, none of which is very pleasant.</P>

<P ALIGN=CENTER><FONT COLOR="#330099" SIZE=+1><HR NOSHADE>&quot;optokinetic
nystagmus&quot;<HR NOSHADE></FONT></P>

<P>The main side effect of optokinetic nystagmus induced by scrolling text
```

```
is a tendency to print the document, so that it can be read later with greater
comfort. This interrupts delivery of content and consumes resources. It
is very possible that the demise of the paperless office dreams of the '80's
was caused in part by nystagmus. Another major cause of its demise was the
difference in formatting between standard paper, height being typically
30% greater than width, and standard computer monitors, width being typically
33% greater than height. Word processors targeted paper formatting, so that
documents delivered from computer to computer were not easily adapted to
screen formats. This pattern also resulted in a tendency to print such documents
rather than struggle to consume their contents on screen.</P>

<P>EZ HTML solves these twin problems nicely. First, there is no scrolling
text, resulting in reduced eye-strain. Font sizing can be adjusted by the
reader for his particular situation of lighting, monitor quality and size,
and his personal visual comfort. Second, the content is automatically laid
out in columns of text, with neither too few words in a line, about 30 characters
minimum, nor too many, about 60 characters maximum. The upper limit yields
a line length which, as the human eye executes its rapid raster motion to
the beginning of the next line, is not too long to accurately and easily
find the correct next new line. If lines are much shorter, below the minimum,
lines of text contain too little content, and the frequent raster motions
can induce eye-strain.</P>

<P ALIGN=CENTER><FONT COLOR="#330099" SIZE=+1><HR NOSHADE>&quot;Web pages
... become a rewarding experience with EZ&nbsp;HTML.&quot;<HR NOSHADE></FONT></P>

<P>This sample document contains about 3300 characters in about 640 words.
At normal reading speeds, it should take about 5 minutes to absorb its main
content. The effort required to read this article on the screen, using EZ
HTML, should be noticeably less than required by other channels. For instance,
using a word processor, two approaches are common. The first is to print
it out, collect the page(s), and finally read it. Alternatively, one could
open the document in a word processor, adjust window widths and zoom factors,
and finally read it, occasionally scrolling to bring unseen parts into view.
With EZ HTML, just sit back, tap the up arrow to adjust the font, and tap
the right arrow to turn the page as needed. Web pages, known widely as challenging
when it comes to delivering on-screen content, can become a rewarding experience
with EZ HTML.</P>

<P><IMG SRC="iwpmdlgo.gif" WIDTH="186" HEIGHT="181" ALIGN="BOTTOM" NATURALSIZEFLAG=
"3"></P>

<P ALIGN=CENTER><IMG SRC="otsdetails.gif" WIDTH="630" HEIGHT="509" NATURALSIZEFLAG=
"3" ALIGN="BOTTOM"></P>

<OL>
  <LI>numbered list 1 level 1 item 1
  <OL>
    <LI>numbered list 2 level 2 item 1
    <LI>numbered list 2 level 2 item 2
    <LI>numbered list 2 level 2 item 3
  </OL>
  <LI>numbered list 1 level 1 item 2
  <OL>
    <LI>numbered list 3 level 2 item 1
    <LI>numbered list 3 level 2 item 2
    <OL>
      <LI>numbered list 4 level 3 item 1
      <OL>
        <LI>numbered list 5 level 4 item 1
        <OL>
          <LI>numbered list 6 level 5 item 1
          <LI>numbered list 6 level 5 item 2
          <LI>numbered list 6 level 5 item 3
        </OL>
        <LI>numbered list 5 level 4 item 2
        <LI>numbered list 5 level 4 item 3
      </OL>
      <LI>numbered list 4 level 3 item 2
      <LI>numbered list 4 level 3 item 3
    </OL>
    <LI>numbered list 3 level 2 item 3
    <LI>numbered list 3 level 2 item 4
  </OL>
  <LI>numbered list 1 level 1 item 3
</OL>

<DL>
  <DT>term 1
  <DD>definition 1 the wordy part of the two part definition item
  <DT>term 2
  <DD>definition 2 the wordy part of the two part definition item
</DL>

<UL>
  <LI>bullet list 1 level 1 item 1
  <UL>
    <LI>bullet list 2 level 2 item 1
    <UL>
      <LI>bullet list 3 level 3 item 1
      <UL>
        <LI>bullet list 4 level 4 item 1
        <LI>bullet list 4 level 4 item 2
        <UL>
          <LI>bullet list 5 level 5 item 1
          <LI>bullet list 5 level 5 item 2
          <LI>bullet list 5 level 5 item 3
        </UL>
        <LI>bullet list 4 level 4 item 3
      </UL>
      <LI>bullet list 4 level 3 item 2
      <LI>bullet list 4 level 3 item 3
    </UL>
    <LI>bullet list 2 level 2 item 2
    <LI>bullet list 2 level 2 item 3
  </UL>
  <LI>bullet list 1 level 1 item 2
  <UL>
    <LI>bullet list 6 level 2 item 1
    <LI>bullet list 6 level 2 item 2
    <UL>
      <LI>bullet list 7 level 3 item 1
      <LI>bullet list 7 level 3 item 2
      <LI>bullet list 7 level 3 item 3
    </UL>
    <LI>bullet list 6 level 2 item 3
  </UL>
  <LI>bullet list 1 level 1 item 3
</UL>

<P>This is normal paragraph text. <B>This is bold text.</B> This is normal
paragraph text. <FONT SIZE=+1>This is big text.</FONT> This is normal paragraph
text. <CITE>This is a citation.</CITE> This is normal paragraph text. <CODE>This
is code text.</CODE> This is normal paragraph text. <EM>This is emphasis
text.</EM> This is normal paragraph text.</P>

<BLOCKQUOTE>
  <P>This is blockquote justified text. <I>This is italic text.</I> This
  is normal paragraph text. <KBD>This is keyboard text.</KBD> This is normal
  paragraph text. <FONT SIZE=-1>This is small text.</FONT> This is normal
  paragraph text <strike>This is strike text.</strike> This is normal paragraph text.
<STRONG>This
  is strong text.</STRONG> This is normal paragraph text. <sub>This is subscript
  text.</sub> This is normal paragraph text. <sup>This is superscript text.</sup>
  This is normal paragraph text.</P>
</BLOCKQUOTE>

<P>This is left justified paragraph text. <TT>This is teletype text.</TT>
This is normal paragraph text. <u>This is underline text.</u> This is normal
paragraph text. <VAR>This is variable text.</VAR> This is normal paragraph
```

```
text.</P>

</BODY>
</HTML>
```